НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і
спеціалізованих комп'ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ В.П. Тарасенко

«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050102 «Комп'ютерна інженерія»

на тему:
«Генератор REST веб-сервісу для баз даних»

Виконав:
студент IV курсу, групи КВ-52
Грицаєнко В.П. _____

Керівник:
Доцент кафедри СПіСКС, к.т.н., доцент,
Тарасенко-Клятченко О.В. _____

Консультант з нормоконтролю:
Доцент кафедри СПіСКС, к.т.н., доцент,
Клятченко Я.М. _____

Рецензент:

_____

_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.050102 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.П. Тарасенко
«___»_____2019 р.

ЗАВДАННЯ

на дипломний проект студенту

Грицаєнку Віктору Павловичу

1. Тема проекту «Генератор REST веб-сервісу для баз даних», керівник проекту Тарасенко-Клятченко Оксана Володимирівна, к.т.н., доц. каф. СПіСКС, затверджені наказом по університету від 22.05.2019 р. №1330-С

2. Термін подання студентом проекту «14» червня 2019 р.

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень генерування REST веб-сервісів та обгрунтування теми; дипломного проекту;
   - визначення моделі «клієнт-сервер»;
   - аналіз архітектурних рішень розробки веб-сервісів;
   - аналіз технологій розробки веб-сервісів та обгрунтування їх використання;
   - розробка генератора веб-сервісів.

5. Перелік обов'язкового графічного матеріалу:
   - UML-діаграма генеруючого веб-сервісу (схема структурна);
   - UML-діаграма основного функціоналу генератора (схема структурна);
   - генерування файлів (схема алгоритму);
   - функціювання графічного інтерфейсу (схема алгоритму).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Доц. каф. СПіСКС Клятченко Я.М.		

7. Дата видачі завдання «__» _____ 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	01.04.2019	Виконано
2.	Розроблення технічного завдання	05.04.2019	Виконано
3.	Розроблення структури проекту генератора	10.04.2019	Виконано
4.	Розроблення функціоналу обробки запитів на створення таблиць	14.04.2019	Виконано
5.	Підготовка матеріалів першого розділу дипломного проекту	24.04.2019	Виконано
6	Розроблення функціоналу генерації файлів веб-сервісу	30.04.2019	Виконано
7.	Підготовка матеріалів другого розділу дипломного проекту	05.05.2019	Виконано
8	Розробка графічного інтерфейсу	10.05.2019	Виконано
9.	Тестування генератора	15.05.2019	Виконано
10.	Підготовка матеріалів третього розділу дипломного проекту	20.05.2019	Виконано
11.	Підготовка матеріалів четвертого розділу дипломного проекту	23.05.2019	Виконано
12.	Підготовка матеріалів п’ятого розділу дипломного проекту	25.05.2019	Виконано
13.	Підготовка графічної частини дипломного проекту	30.05.2019	Виконано
14.	Оформлення документації дипломного проекту	02.06.2019	Виконано

Студент _____ Грицаєнко В.П.
(підпис)

Керівник проекту _____ Тарасенко-Клятченко О.В.
(підпис)

АНОТАЦІЯ

Об’єкт розробки – генератор REST веб-сервісу для баз даних. Програмний комплекс дозволяє:

- генерувати серверний Java-проект, в основі якого лежать фреймворки Spring Boot та Hibernate з можливістю конфігурації генеруючого проекту;
- налаштовувати та програмно розширювати код генератора;
- налаштовувати підключення до баз даних для генеруючого проекту;
- створити таблиці в базі даних, з якими буде працювати генеруючий веб-сервіс;
- налаштовувати валідацію вхідних на сервер даних.

Результатом є згенерований веб-сервіс, що може бути налаштований та запущений на локальній машині користувача. Після запуску користувач має змогу: надсилати запити на сервер для зберігання, діставання, оновлення та видалення даних з таблиць бази даних; переглянути можливості згенерованого веб-сервісу на відповідній веб-сторінці та відсилати запити за її допомогою.

В ході розробки:

- проведено аналіз методів розробки існуючих генераторів коду;
- сформульовані вимоги до генератора REST веб-сервісу;
- розроблений функціонал для зчитування SQL-запитів різних діалектів для визначення структури таблиць бази даних;
- розроблений функціонал для генерування файлів проекту;
- розроблений функціонал для налаштування з’єднання різними типами баз даних;
- розроблений користувацький інтерфейс.

Використання даного програмного комплексу дозволить створити та налаштувати стандартний Java проект REST веб-сервісу, зменшити час написання шаблонного коду. Користувачі можуть використовувати згенерований проект для базової роботи з REST веб-сервісом, а розробники можуть використовувати згенерований проект як відправну точку для розробки свого програмного забезпечення.

Ключові слова:

JAVA, ГЕНЕРАТОР КОДУ, REST API, SPRING BOOT, HIBERNATE, SQL, ВЕБ-СЕРВІС

ABSTRACT

The object of development is a REST web-service generator for databases.

The application allows to:

- generate a server-side Java project build on Spring Boot and Hibernate frameworks with ability to configure generating project;
- configure and programmatically extend the code of the generator;
- configure database connection for the generating project;
- create tables in the database the generating web service will work with;
- configure request data validation.

The result is the generated web service which can be configured and run on local user's computer. While the web service is running user have an ability to: send requests to the server to store, fetch, refresh and delete data from the database; see generated web service functional using special web page and use it to send request to the server;

During the development of the diploma project:

- approaches of the code generation were analyzed;
- REST web-service generator requirements were set forth;
- SQL queries parsing functional was created for various dialects;
- project file generation functional was created;
- request input data validation functional was created;
- database connection configuration functional was created;
- user interface was developed.

Using the generator users may create and configure REST web service java project. It allows decrease time of writing boilerplate code. Users have ability to use generated project to work with generated REST web service. Developers may use it as a start point for their own software development.

Ключові слова:

JAVA, CODE GENERATOR, REST API, SPRING BOOT, HIBERNATE, SQL, WEB-SERVICE

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація загальна			
			Новорозроблена			
	А4	ІАЛЦ. 045490.002 ТЗ	Генератор REST	4		
			веб-сервісу			
			Технічне завдання			
	А4	ІАЛЦ. 045490.003 ТП	Генератор REST	1		
			веб-сервісу			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ. 045490.004 ПЗ	Генератор REST	60		
			веб-сервісу			
			Пояснювальна записка			
	А4	ІАЛЦ. 045490.005 Д1	Генератор REST	1		
			веб-сервісу			
			UML-діаграма			
			генеруючого веб-сервісу			
			Схема структурна			
	А4	ІАЛЦ. 045490.006 Д2	Генератор REST	1		
			веб-сервісу			
			UML-діаграма основного			

Зм	Лист	№ докум.	Підп	Дата	ІАЛЦ. 045490.001 ОА		
Розроб.		Грицаєнко В.П.			Генератор REST веб-сервісу для баз даних *Опис альбому*	Літ.	Лист 1 / Листів 2
Перев.		Тарасенко- -Клятченко О.В.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-52	
Н. контр.		Клятченко Я.М.					
Затв.		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			функціоналу генератора			
			Схема структурна			
	А4	ІАЛЦ. 045490.007 Д3	Генератор REST	1		
			веб-сервісу			
			Генерування файлів			
			проекту			
			Схема алгоритму			
	А4	ІАЛЦ. 045490.008 Д4	Генератор REST	1		
			веб-сервісу			
			Функціювання графічного			
			інтерфейсу			
			Схема алгоритму			
		Диск CD-ROM	Текст ПЗ. Тексти програм.	1		
			Графічний матеріал.			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Генератор REST веб-сервісів для баз даних».

Галузь застосування: розробники, користувачі та підприємства, що потребують REST веб-сервіс для роботи з базами даних.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи ступеня «бакалавр комп'ютерної інженерії», затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення програмного генератора Java Spring Boot REST веб-сервісу за заданою конфігурацією.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- Сумісність з будь якою операційною системою, що має встановлений JDK (Java Development Kit);
- Можливість генерування серверного Java проекту на Spring Boot фреймворку та системі автоматочного збирання Maven або Gradle;

- Можливість задання структури таблиць в базі даних за допомогою SQL-запитів на різних SQL-діалектах;
- Можливість задання валідації вхідних на веб-сервіс параметрів;
- Можливість налаштування підключення до різних типів баз даних;
- Можливість гнучкого налаштування генератора;
- Можливість гнучкого розширення API генератора;
- Можливість розширення згенерованого коду;
- Генерування проекту, побудованого згідно останніх найкращих практик та бібліотек, фреймворків;
- Можливість компіляції та запуску згенерованого веб-сервісу;
- Наявність користувацького інтерфейсу.

5.2. Вимоги до апаратного забезпечення

- Процесор: 2-х ядерний, Intel, AMD;
- Оперативна пам’ять: 2 Гб;
- Наявність доступу до мережі Internet;

5.3. Вимоги до програмного та апаратного забезпечення користувача

- Операційна система Windows, Unix-подібні системи;
- Наявність встановленного та налаштованого JDK (Java Development Kit);
- Наявність встановленої та налаштованої системи автоматичного збирання Maven або Gradle.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	01.04.2019	
2.	Розроблення технічного завдання	05.04.2019	
3.	Розроблення структури проекту генератора	10.04.2019	
4.	Розроблення функціоналу обробки запитів на створення таблиць	14.04.2019	
5.	Підготовка матеріалів першого розділу дипломного проекту	24.04.2019	
6	Розроблення функціоналу генерації файлів веб-сервісу	30.04.2019	
7.	Підготовка матеріалів другого розділу дипломного проекту	05.05.2019	
8	Розробка графічного інтерфейсу	10.05.2019	
9.	Тестування генератора	15.05.2019	
10.	Підготовка матеріалів третього розділу дипломного проекту	20.05.2019	
11.	Підготовка матеріалів четвертого розділу дипломного проекту	23.05.2019	
12.	Підготовка матеріалів п’ятого розділу дипломного проекту	25.05.2019	
13.	Підготовка графічної частини дипломного проекту	30.05.2019	
14.	Оформлення документації дипломного проекту	02.06.2019	

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			Документація проекту			
	А4	ІАЛЦ. 045490.004 ПЗ	Генератор REST	60		
			веб-сервісу			
			Пояснювальна записка			
	А4	ІАЛЦ. 045490.005 Д1	Генератор REST	1		
			веб-сервісу			
			UML-діаграма			
			генеруючого веб-сервісу			
			Схема структурна			
	А4	ІАЛЦ. 045490.006 Д2	Генератор REST	1		
			веб-сервісу			
			UML-діаграма основного			
			функціоналу генератора			
			Схема структурна			
	А4	ІАЛЦ. 045490.007 Д3	Генератор REST	1		
			веб-сервісу			
			Генерування файлів			
			проекту			
			Схема алгоритму			
	А4	ІАЛЦ. 045490.008 Д4	Генератор REST	1		

					ІАЛЦ. 045490.003 ТП
Зм	***Лист***	***№ докум.***	***Підп***	***Дата***	
Розроб.		Грицаєнко В.П.			Генератор REST веб-сервісу для баз даних ***Відомість технічного проекту***
Перев.		Тарасенко- -Клятченко О.В.			
Н. контр.		Клятченко Я.М.			
Затв.		Тарасенко В.П.			

Літ.	***Лист***	***Листів***
	1	2

КПІ ім. Ігоря Сікорського, ФПМ, КВ-52

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
			веб-сервісу			
			Функціювання графічного			
			інтерфейсу			
			Схема алгоритму			
		Диск CD-ROM	Текст ПЗ. Тексти програм.			
			Графічний матеріал.			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

БД – база даних;

Бібліотека підпрограм – збірка об’єктів чи програм для вирішення близьких за тематикою задач;

Веб-сервіс – програмна система, що ідентифікується URI, має публічні інтерфейси та може використовуватися іншими додатками у мережі;

ПЗ – програмне забезпечення;

Програмний проект – набір файлів програмного коду для рішення певної задачі, що має певну структуру та конфігурацію;

Фреймворк - інфраструктура програмних рішень, що полегшує розробку складних систем;

CORS - Cross-Origin Resource Sharing – механізм для конфігурації безпеки додатків, що дозволяє клієнту отримувати доступ тільки до визначених ресурсів;

Hibernate - фреймворк, що дозволяє працювати з БД за допомогою Java-коду та представляти SQL-таблиці у вигляді Java-об’єктів;

HTTP – Hypertext Transfer Protocol – протокол прикладного рівня для передачі даних;

IDE – Integrated Development Environment – система програмних засобів, що використовується для розробки ПЗ;

Java – сильно типізована об’єктно-орієнтовна мова програмування, розроблена компанією «Sun Microsystems»;

JSON – JavaScript Object Notation – текстовий формат обміну даними;

REST - Representational State Transfer – архітектурний стиль взаємодії компонентів розподілений елементів додатку в мережі;

Spring – відкритий універсальний програмний каркас для розробки додатків мовою Java;

Spring Boot – програмний каркас для спрощення створення ПЗ на основі фреймворку Spring;

URI - Uniform Resource Identifier - компактний рядок літер, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс;

URL – Uniform Resource Locator – стандартизована адреса певного ресурсу.

ВСТУП

Сучасні мови програмування та підходи до розробки програмного забезпечення ставлять собі в мету якомога збільшити швидкість розробки ПЗ, зробити його модульним, а модулі – незалежними. Це робить можливим підтримку та розширення коду ПЗ. Вимоги досягаються шляхом реалізації простого та зрозумілого синтаксису мов програмування, використання додаткових фреймворків, технологій та методологій розробки ПЗ тощо.

Одним із підходів, що допомагає реалізувати вимоги наведені вище, є генерування коду. В деяких випадках підходи до реалізації певних загальних питань є шаблонними – тобто можуть реалізовані автоматично. Для генерування коду можуть використовуватися як сторонні бібліотеки, так і самостійні програми. Основне використання мають саме сторонні бібліотеки, які використовуються під час розробки ПЗ для генерування самостійних компонентів програмного коду. Однак метою даної роботи є самостійна програма, яка не буде вимагати користувача знати технічні аспекти програмування на мові Java, а, маючи встановлене ПЗ для функціювання Java та налаштоване підключення БД, генерувати та запускати веб-сервіс, з яким можна буде обмінюватися повідомлення в мережі Інтернет за допомогою REST-запитів.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ГЕНЕРУВАННЯ REST ВЕБ-СЕРВІСІВ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1 Аналіз способів генерування коду

Задача генерування коду полягає у створенні робочого коду за заданими параметрами для автоматизації шаблонних процесів.

Існують наступні підходи до генерування програмного коду:

1. Генерування коду рядок за рядком безпосередньо з програмного коду генератора;
2. Генерування коду за заданим шаблоном.

Генерування коду рядок за рядком безпосередньо з програмного коду генератора

При генеруванні коду безпосередньо з програмного коду генератора кожен компонент генерується окремо, використовуючи тимчасовий буфер, вміст якого записується в результуючий файл. Використання буферу дозволить зменшити навантаження на фізичний носій інформації. Структура, вміст параметри та форматування результуючого коду задається в коді його генератора. Деякі параметри можуть бути винесені в конфігураційний файл або БД та зчитуватися під час генерування.

Переваги:

- простота і швидкість реалізації;
- відсутність потреби використання сторонніх бібліотек, або використання легковісних.

Недоліки:

- складність або неможливість генерування складних структур;
- складність у підтримці та модифікуванні коду генератора;
- один компонент генератора відповідальний за велику кількість різноманітної роботи;

- Незручність форматування генеруючого коду.

Генерування коду за заданим шаблоном

Для генерування коду за заданим шаблоном використовують так звані Template Engine, тобто движки генерування текстових файлів за шаблоном. Шаблон є зовнішнім ресурсом у вигляді файлу, який містить статичні дані та опис даних, які підставляються динамічно з визначених раніше назв на значень змінних. Збірка змінних, що використовується в шаблоні, називається контекстом шаблону. Контекст дозволяє визначати динамічні значення в результуючому файлі, які можуть залежати від вводу користувача, конфігурації тощо. При генеруванні движок ідентифікує змінні в шаблоні, знаходить їх значення в контексті та замінює назви останніх на їх безпосередні значення.

Зазвичай шаблонні движки підтримують базові управляючі конструкції (оператори розгалуження, циклів), дають змогу створювати та повторно використовувати створені змінні в шаблонах, підтримують форматування.

Переваги:

- відокремлення шаблону та його форматування від логіки генерування;
- шаблони являють собою зовнішні файлі ресурсів, які є читабельними і зрозумілими для користувача;
- вміст шаблонів може бути змінений без необхідності збірки та компіляції коду генератора.

Недоліки:

- необхідність використання сторонніх бібліотек;
- назви змінних в шаблоні залежать від назв, визначених у програмному коді. Зміна назв змінних, структури об’єктів в контексті вимагає відповідної їх зміни у файлах шаблонів.

1.2 Аналіз існуючих рішень з генерування коду REST веб-сервісів

Генерування коду є важливим аспектом для розробки ПЗ мовою Java. Існуючі генератори коду переслідують різні цілі та мають різні підходи до реалізації. Для генерування коду веб-сервісів можуть бути використані наступні бібліотеки та утиліти:

- JavaPoet
- Spring Roo
- FastCode Eclipse Plugin
- RAML

JavaPoet

Деякі генератори коду використовуються для генерування окремих Java-класів. Прикладом такої бібліотеки є JavaPoet. Дана бібліотека дозволяє генерувати Java-класи безпосередньо з Java-коду, що може бути корисним для більшості розробників. Кожна базова структура мови Java може бути описана викликом відповідного метода.

Переваги:

- можливість гнучкої побудови Java-коду рядок за рядком;
- бібліотека дбає про форматування, генерування та збереження файлів.

Недоліки:

- Java-код генерується безпосередньо з Java-коду, що вимагає користувача досконально знати синтаксис, підходи до програмування мовою Java та вміти використовувати дану бібліотеку;
- інструмент дозволяє генерувати код з базовими конструкціями мови, що є корисним для генерування простого коду різними підходами, але є незручним для генерування і підтримки великого Java-проекту;

- бібліотека не дає можливості встановити залежність генеруючого Java-класу з іншими генерованими таким методом Java-класами;
- складність або неможливість генерування повноцінного налаштованого робочого проекту.

Spring ROO

Spring ROO використовується для генерування коду REST веб-сервісів. Кожен REST веб-сервіс має стандартну структуру, яка складається з наступних компонентів:

1. Об’єкт – інкапсулює перелік полів, які однозначно визначають рядки в таблиці БД;
2. Сервіс - виконує основну логіку по обробці даних;
3. DAO, або репозиторій – виконує роль доступу до БД;
4. Контролер – визначає інтерфейс для «спілкування» з іншими сервісами в мережі Інтернет.

Дана утиліта використовує свою CLI для взаємодії з користувачем та генерує один з вище перерахованих компонентів REST веб-сервісу за допомогою одної команди або декількох команд. Для генерування кожного рядка об’єкту, та, відповідно, рядка в БД використовується окрема команда, яка задає його тип, назву та властивості.

Переваги:

- утиліта генерує повноцінний налаштований Java-проект на базі фреймворку Spring Boot зі всіма необхідними залежностями;
- задає визначену структуру проекту;
- має можливість налаштування кожного з компонентів REST веб-сервісу окремо;
- згенерований код може бути розширений, модифікований та використаний в майбутньому.

Недоліки:

- вміст генеруючих файлів однозначно визначений та не може бути змінений за бажанням;
- потребує розуміння структури REST веб-сервісу з точки зору розробника.

FastCode Eclipse Plugin

FastCode – плагін для Eclipse IDE, що генерує довільний Java-код за заданим шаблоном. Плагін має свій користувацький інтерфейс, дозволяє створювати, зберігати та використовувати існуючі шаблони для генерування коду. Утиліта може ефективно використовуватися розробниками для генерування шаблонного коду та збільшити швидкість написання ПЗ.

Переваги:

- універсальність, дозволяє генерувати довільний Java-код;
- зручне управління шаблонами;
- гнучкість конфігурації.

Недоліки:

- велика кількість параметрів конфігурації;
- потребує розуміння структури генеруючого коду з точки зору розробника;
- не має змоги генерувати повноцінні Java-проекти зі всіма необхідними залежностями.

RAML

RAML являє собою генератор, структура веб-сервісу для якого задається за допомогою спеціального .raml файлу за допомогою спеціального синтаксису, що дозволяє задавати розташування ресурсів, типи даних, валідацію параметрів, тип відповіді, що повертає ресурс тощо.

Переваги:

- гнучкість конфігурації;
- можливість генерування документації.

Недоліки:

- велика кількість параметрів конфігурації;
- потребує знання синтаксису конфігураційних файлів та розуміння конфігурації веб-сервісів в цілому.

1.3 Обґрунтування теми дипломного проекту

Метою даної роботи є розробка генератора REST веб-сервісу, який:

- не вимагає від користувача знання спеціального синтаксису, структури проекту, деталей реалізації тощо;
- генерує Java-код, що побудований на сучасних та ефективних бібліотеках, фреймворках та архітектурних рішеннях;
- має зручну конфігурацію та функціонал якого може бути гнучко розширений іншими розробниками;
- може використовуватись як розробниками для ініціалізації базового проекту, так і користувачами, які працюють з мовою SQL та мають налаштоване підключення до БД;
- має графічний інтерфейс взаємодії з користувачем, що дозволяє зручним чином ввести параметри генерування проекту та запобігти некоректного вводу зі сторони користувача.

Користувач може задати структуру БД SQL-запитами на створення SQL-таблиць, задати параметри підключення до БД, перевірити з’єднання з останньою, зібрати та запустити згенерований проект. Тобто, користувач працює саме з БД, та не потребує технічні навички розробки REST веб-сервісів, їх збірки та запуску.

1.4 Обґрунтування вибору мови програмування

Мова програмування Java є однією із самих використовуваних та зручних мов програмування для розробки веб-сервісів. Мова має відносно

простий синтаксис, сильну типізацію. Java-проекти є легкими у підтримці, для даної мови програмування існують ефективні та зручні компоненти для розробки ПЗ.

Для даної мови програмування розроблені ефективні системи автоматизації збірки проектів, такі як Maven та Gradle, які дозволять в одну команду зібрати та запустити згенерований проект. Фреймворки Spring Boot та Hibernate дозволяють зменшити кількість генеруючого коду, підвищити його простоту та надійність, задати базову структуру проекту.

2. ВИЗНАЧЕННЯ МОДЕЛІ «КЛІЄНТ-СЕРВЕР»

2.1 Компоненти веб-сервісу

При реалізації ПЗ кожен його компонент виконує певні функції. Деякі компоненти виконують функцію взаємодії з користувачем (зчитування команд від користувача, відображення результату виконання програми), деякі виконують визначену прикладку логіку, інші володіють та керують ресурсами (зберігають, оновлюють та дістають дані).

В сучасних підходах до реалізації програмного забезпечення функції програмних засобів можна поділити на 3 умовних типи:

1. Функції введення та відображення інформації;
2. Прикладні функції, що виконують визначену логіку спеціальної області;
3. Функції, що володіють ресурсами (даними).

Відповідно компоненти програми можна поділити на три умовних типи згідно функцій, які вони виконують:

1. Компоненти, що виконують функцію взаємодії з користувачем;
2. Компоненти, що виконують основну логіку програми, реалізуючи певну послідовність дій;
3. Компоненти, що керують ресурсами.

Комп’ютери або програми, що володіють та керують ресурсами, називають серверами, а компоненти, що виконують представлення даних користувачу – клієнтами. Компоненти або програми, що виконують основну логіку програми називають прикладними.

Всі компоненти є незалежні між собою та кожен виконує свою визначену функцію. Задача полягає в налагоджені взаємодії між ними. Кожен веб-сервіс складається з компонентів, представлених на рисунку 2.1.

Рисунок 2.1 - Компоненти веб-сервісу

Модель «клієнт-сервер» визначає загальні принципи організації взаємодії в мережі, де є сервери – вузли-постачальники деяких специфічних функцій та клієнти – споживачі цих функцій.

Практичні реалізації такої архітектури називаються клієнт-серверними технологіями. Кожна технологія визначає власні або використовує наявні правила взаємодії між клієнтом і сервером, які називаються протоколом обміну (протоколом взаємодії).

2.2 Дворівнева модель «Клієнт-Сервер»

Більшість мережевих застосунків так чи інакше реалізовані за допомогою архітектурного рішення «Клієнт-Сервер». Для реалізації простих мережевих застосунків достатньо два компоненти – клієнта, що відповідальний за відображення даних та сервера, що керує даними. За наявності в системі двох компонентів таке архітектурне рішення називають дворівневим.

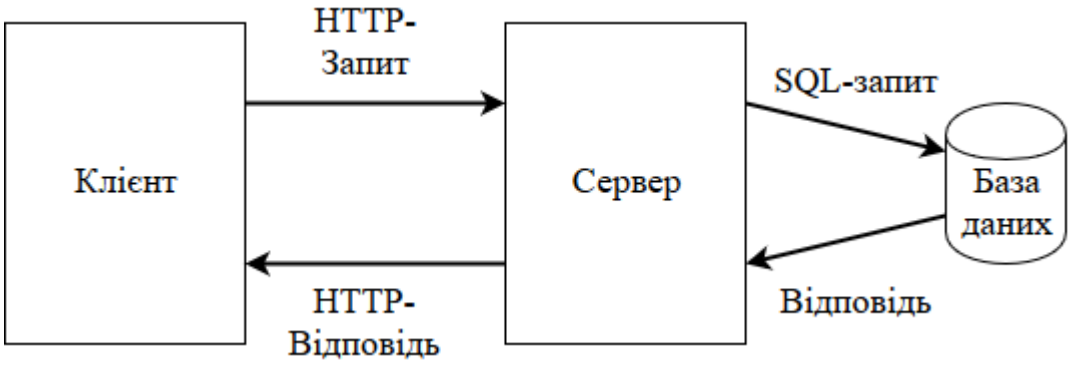

Рисунок 2.2 - Дворівнева архітектура «Клієнт-Сервер»

Дана схема використовується для реалізації мережевих застосунків. В даному випадку сервер безпосередньо виконує клієнтські запити, не звертаючись до інших серверів чи ресурсів.

В залежності від необхідного функціоналу, БД в системі може бути відсутня. Сервер може бути відповідальний тільки за розрахунки (прикладні дії), а не збереження та управління ресурсами, або за виконання обох функції.

Переваги:

- простота реалізації та налаштування;
- висока швидкодія (немає необхідності звертатися до інших ресурсів).

Недоліки:

- складність масштабування (один сервер може бути відповідальним як за управління ресурсами, так і виконання прикладних дій).

2.3 Трирівнева модель «Клієнт-Сервер»

Для спрощення масштабування та розділення відповідальності в дворівневу архітектуру доцільно ввести ще один компонент, який буде відповідальний за виконання прикладних дій. Таким компонентом є прикладний сервер, а архітектура має назву трирівневої.

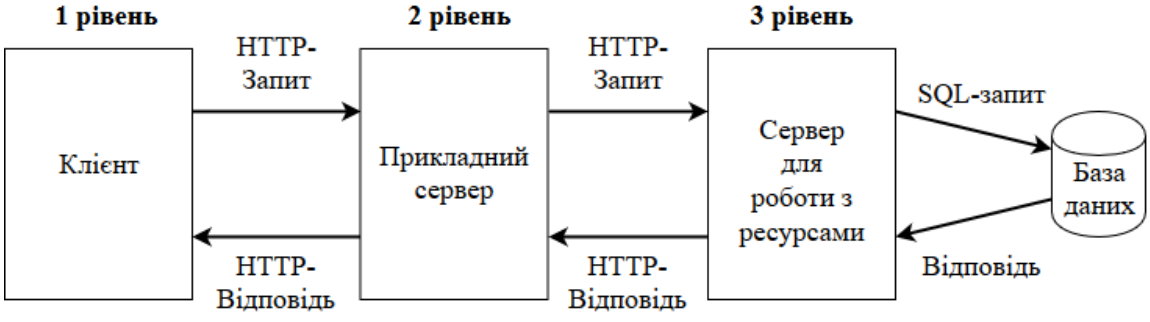

Рисунок 2.3 - Трирівнева архітектура «Клієнт-Сервер»

Прикладний сервер дозволяє ввести логіку розподілених обчислень в систему та використовується в якості проміжного програмного забезпечення.

Переваги:

- спрощення масштабування;
- розподілення відповідальності між серверами;
- кожен рівень є незалежним, кожен сервер може бути запущений окремо на різних платформах;
- менші потреби до апаратних ресурсів для роботи окремих серверів;
- краща захищеність системи.

Недоліки:

- складність узгодження рівнів між собою;

2.4 Багаторівнева модель «Клієнт-Сервер»

Для більшого розділення відповідальності до трирівневої структури можна додати довільну кількість нових рівнів, де кожен окремий рівень є незалежним та виконує визначену логіку. Кожен рівень реалізує сукупність спільних задач, які є взаємозв’язаними та можуть бути відокремлені від інших рівнів.

Багаторівнева архітектура посилює переваги трирівневої, але у багаторівневої є важливий недолік – складність конфігурації та узгодження компонентів в системі.

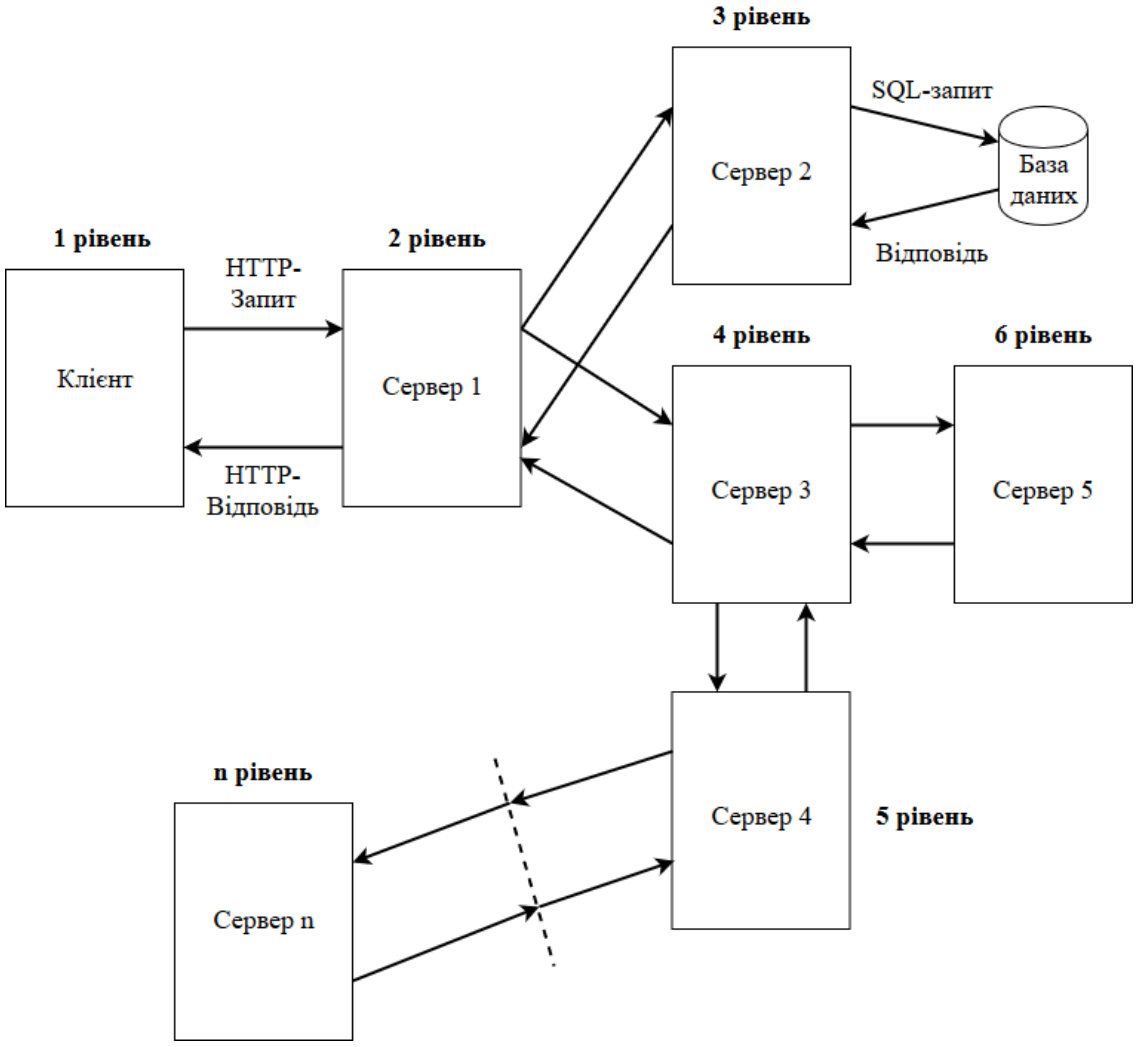

Рисунок 2.4 - Багаторівнева архітектура «Клієнт-Сервер»

2.5 Мета дипломного проекту стосовно моделі «Клієнт-Сервер»

Метою генератора REST веб-сервісу є генерування та налаштування серверу для роботи з БД заданої структури. Такий сервер позначено як «рівень 2» на рисунку 2.2, або як «рівень 3» на рисунку 2.3. Важливо зазначити, що у випадку дворівневої архітектури сервер може бути відповідальним за виконання прикладної логіки. Так як остання є специфічною для кожної області, генерування прикладної логікі не входить в мету дипломного проекту. Теж саме стосується клієнтського рівня, хоча генерування простого веб-інтерфейсу є метою дипломного проекту. Останній виконує роль документації

та дозволяє відсилати HTTP-запити на згенерований сервер, але не є повноцінним клієнтським інтерфейсом.

Реалізація серверу для роботи з БД в більшій мірі є шаблонною, тому може бути автоматизована. Спрощено, інтерфейс серверу визначається структурою БД та архітектурними шаблонами.

3. АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ РОЗРОБКИ ВЕБ-СЕРВІСІВ

3.1 Поняття веб-сервісу

Веб-сервіс, або веб-служба – програмний комплекс, що ідентифікується URI, виконує визначені функції, має публічні інтерфейси та може використовуватися іншими компонентами у мережі.

Веб-сервіси є незалежними, забезпечують взаємодію компонентів у мережі незалежно від платформи. Веб-сервіси базуються на відкритих стандартах та протоколах, що визначають їх інтерфейс та спосіб взаємодії з іншими компонентами у мережі. Кожен веб-сервіс виконує свою прикладну програму.

3.2 Поняття інструменту автоматичної збірки проектів

Автоматична збірка або автоматична побудова – етап розробки програмного забезпечення, суть якого полягає в автоматизації наступного спектру задач:

- Компіляція програмного коду;
- Збірка бінарного коду в виконуваний файл;
- Виконання тестів;
- Розгортання програми на вибраній платформі;
- Запуск генерування
- коду.

Автоматична збірка має велику кількість переваг, а саме:

- поліпшення якості продукту;
- автоматизація шаблонних процесів збірки програмного коду;
- позбавленні розробників від зайвих дій;
- мінімізація помилок;

- ведення історії складань;
- економія часу та фінансів.

Найбільш поширеними інструментами збірки проектів, розроблених мовою Java, є Maven та Gradle. Кожна система має файл конфігурації, в якому можна визначити плагіни, що виконують необхідні дії, залежності проекту, параметри проекту, репозиторії, з яких завантажувати вказані залежності тощо. Такий файл розміщується в корені проекту. Maven використовує файл під назвою pom.xml, Gradle – build.gradle. Ці назви є зарезервованими та не можуть бути зміненими.

3.3 Архітектурний шаблон MVC

REST веб-сервіс на мові Java доцільно реалізовувати за архітектурним шаблоном MVC (або модель-вигляд-контролер, англ. Model-View-Controller). Шаблон передбачає реалізацію трьох взаємопов’язаних компонентів:

1. Модель – абстрактне представлення даних, що відображують предмети реального світу;
2. Вигляд – реалізація представлення даних користувачу. Користувацький інтерфейс, за допомогою якого користувач може взаємодіяти з програмою;
3. Контролер – реагує на дії користувача, виконуючи або делегуючи виконання визначених дій.

Шаблон дозволяє відокремити логічні компоненти програми, полегшити їх зміни та розширення. Підхід є часто вживаним при розробці програмного забезпечення різного призначення. Тобто, для розробника такий підхід є зрозумілим та дозволяє швидко зрозуміти логіку програмного комплексу.

Переваги шаблону MVC:

- модулі програми незалежні;

- окремі модулі можуть бути легко замінені іншими на необхідності;
- проста інтеграція нових реалізації модулів.

Шаблон MVC не має явних недоліків, але такі можуть з’явитися при нерозумінні суті шаблону та недотриманні правил та підходів до реалізації.

3.4 Архітектурний стиль взаємодії компонентів в мережі REST

REST (або передача стану представлення, англ. Representational State Transfer) – архітектурний стиль взаємодії компонентів в мережі. REST не є визначеним архітектурним шаблоном, а являє собою набір узгоджених правил та обмежень, що накладаються на мережевий застосунок. Мережеві застосунки, що відповідають правилам та обмеженням REST, називають “RESTful”.

В мережі REST-запит являє собою звичайний HTTP-запит. REST визначає використання наступних типів HTTP-запитів:

1. GET (отримання ресурсу);
2. POST (створення ресурсу);
3. PUT (оновлення ресурсу);
4. DELETE (видалення ресурсу).

Для HTTP-протоколу існує ряд службових типів запитів, але останні не використовуються в RESTful додатках за відсутності практичної потреби.

Для обміну повідомленнями з REST веб-сервісами прийнято використовувати тестовий формат передачі даних під назвою JSON. За його допомогою можна представити дані, що передаються REST веб-сервісу або віддаються з нього. Дані, передані за допомогою JSON, мають вигляд:

```
{
    “firstName”: “Іван”,
    “lastName”: “Іванов”
```

}

Де зліва вказані назви параметрів у лапках, а після двокрапки – їх значення. Такий формат є зручним для читання та розуміння та, маючи простий синтаксис, займає мінімальний розмір, дозволяючи зменшити результуючий трафік.

На веб-сервіси, реалізовані за допомогою архітектурного стилю REST, накладаються наступні основні вимоги:

1. Використання моделі «Клієнт-Сервер»;
2. Відсутність внутрішнього стану (сесії);
3. Одноманітність інтерфейсу;
4. Структури програмної реалізації розбита по рівням.

Під відсутністю внутрішнього стану розуміється відсутність будь-якої сесії, що зберігає інформацію між запитами. При використанні такого підходу виконання запитів веб-сервісом стає простим та ізольованим від інших запитів, що дозволяє спростити реалізацію та підтримку веб-сервісу з програмної точки зору.

Одноманітність інтерфейсу визначає типи запитів, які може приймати REST веб-сервіс, їх назви, підходи реалізації на мету. Одноманітність структури реалізації веб-сервісу визначає стандартну структуру проекту, розбиту по незалежним рівням, кожен з яких виконує свою роль та використовується на рівні вище. Такий підхід дозволяє спростити структуру програмної реалізації веб-сервісу, зробити розробку та підтримку веб-сервісу простішою.

Переваги використання архітектурного стилю REST:

- одноманітність програмного інтерфейсу та структури веб-сервісу;
- швидкодія роботи;
- простота масштабування;
- надійність системи;

- прозорість зв’язків між компонентами системи;
- можливість ПЗ використання різними клієнтами.

3.5 Структура REST веб-сервісу

Реалізація веб-сервісів на мові Java передбачає собою визначену структуру програмного комплексу. На рисунку 3.1 показані компоненти веб-сервісу, побудованого за шаблоном MVC. В ній наявний Контролер та Модель, визначені вище. При реалізації веб-сервісу доцільно визначити ще дві сутності – сервіс та DAO (або об’єкт доступу до даних, англ. Data Access Object). Сервіс необхідний для виконання основної логіки програми. Сервіс маніпулює моделлю (даними), робить розрахунки, звертається до інших компонентів програми. Об’єкт доступу до даних використовується для безпосереднього доступу до БД та виконання запитів до останньої.

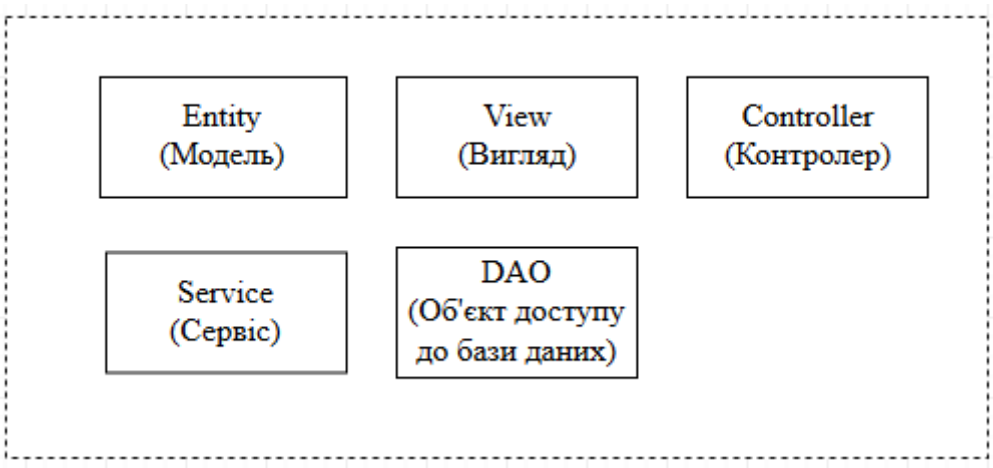

Рисунок 3.1 - Компоненти веб-сервісу, реалізованого за шаблоном MVC

Вигляд (View) з шаблону MVC може мати різні реалізації: користувацький графічний інтерфейс, інтерфейс командної строки, html-сторінка тощо. Але метою розробки цього проекту є генератор REST веб-сервісів, що являють собою інтерфейс для доступу до БД та виконання основних шаблонних операцій: отримання, додавання, видалення та

оновлення даних. Тобто генерування серверної частини для роботи з БД. Користувацький інтерфейс може бути реалізований користувачем за необхідності. Більше того, один згенерований веб-сервіс може використовуватися багатьма користувацькими інтерфейсами.

Важливим зауваженням є те, що REST веб-сервіс не може мати реалізації Вигляду (View) за своїм визначенням.

Для реалізації повноцінного веб-сервісу доцільним є визначити декілька додаткових компонентів. Їх реалізація дозволить зробити програмний комплекс ще більш модульним, де кожен компонент відповідальний за свою невелику частину роботи. Такими компонентами є:

- представлення запиту на веб-сервіс (Request);
- представлення відповіді веб-сервісу (Response);
- компонент, що відповідальний за перетворення запитів та відповідей в Модель (Mapper).

Введення вищеперелічених компонентів дозволить змінювати вміст серверних запитів та відповідей, тобто визначати вміст JSON’у, за допомогою якого веб-сервіс обмінюється повідомленнями з клієнтами та іншими компонентами в мережі. Це необхідно, коли в Моделі деякі моделі визначаються не зі змісту запита, а, наприклад, розраховуються, дістаються з інших компонентів тощо. Прикладом таких параметрів є теперішня дата та час, що можуть визначатися на сервері. Тобто, відсилати їх в запиті на сервер не є доцільним.

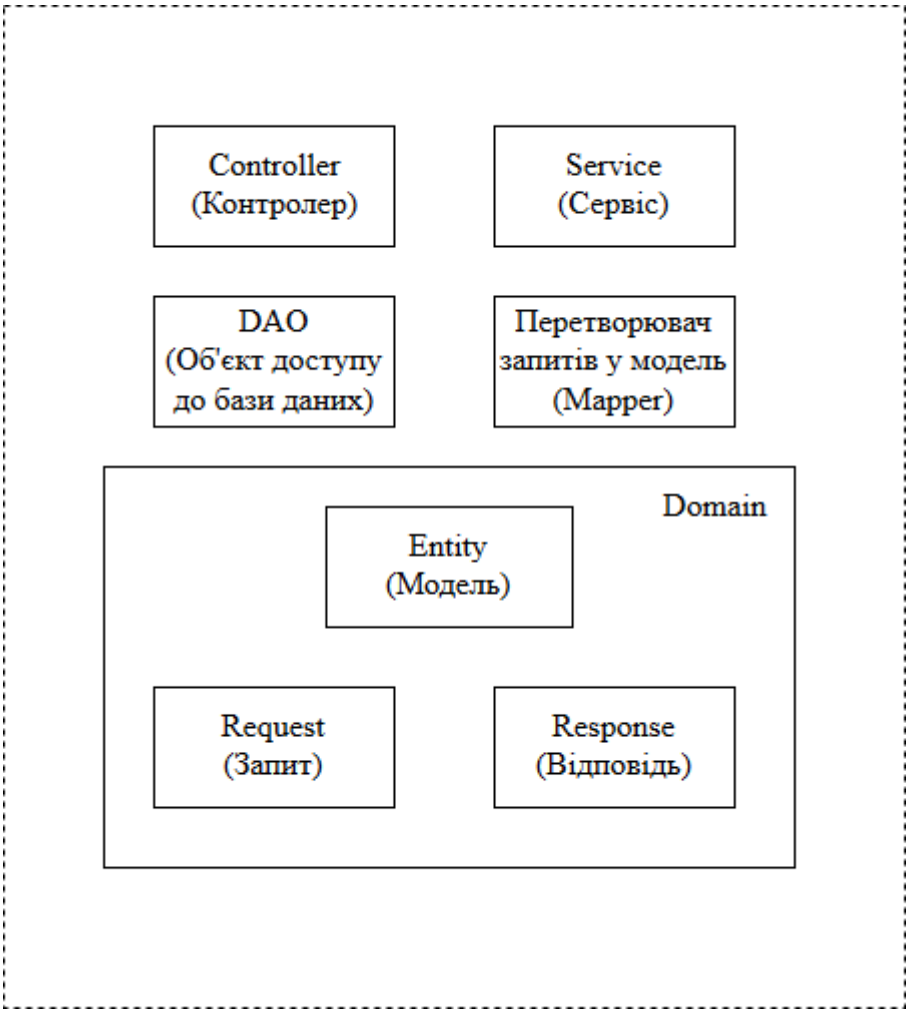

Рисунок 3.2 - Структурна схема REST веб-сервісу

Також важливою частиною додатку є його конфігурація. Умовно можна визначити конфігурацію додатку та конфігурацію безпеки додатку.

Конфігурація додатку може містити в собі визначення його компонентів та параметрів. Назвемо її Конфігурацією додатку (AppConfig).

Безпека веб-сервісу визначає можливість доступу до нього іншими додатками. Захищеність веб-сервісу налаштовується переліком можливих ресурсів у мережі, що можуть мати доступ до нього, використання логіну чи паролю для доступу до ресурсу, спеціальних токенів тощо. Визначимо конфігурацію безпеки додатку під назвою Конфігурація безпеки (SecurityConfig).

При роботі веб-сервісу можуть виникати помилки, при виникненні яких неможливо продовжувати подальшу роботу. Необхідним є визначення всіх можливих помилок в додатку, їх обробка та сповіщення користувача конструктивним та інформативним повідомленням. Для цього до системи необхідно додати поняття Помилки (Exception) та Оброблювача помилок (ExceptionHandler).

Як результат, маємо розширену структуру веб-сервісу:

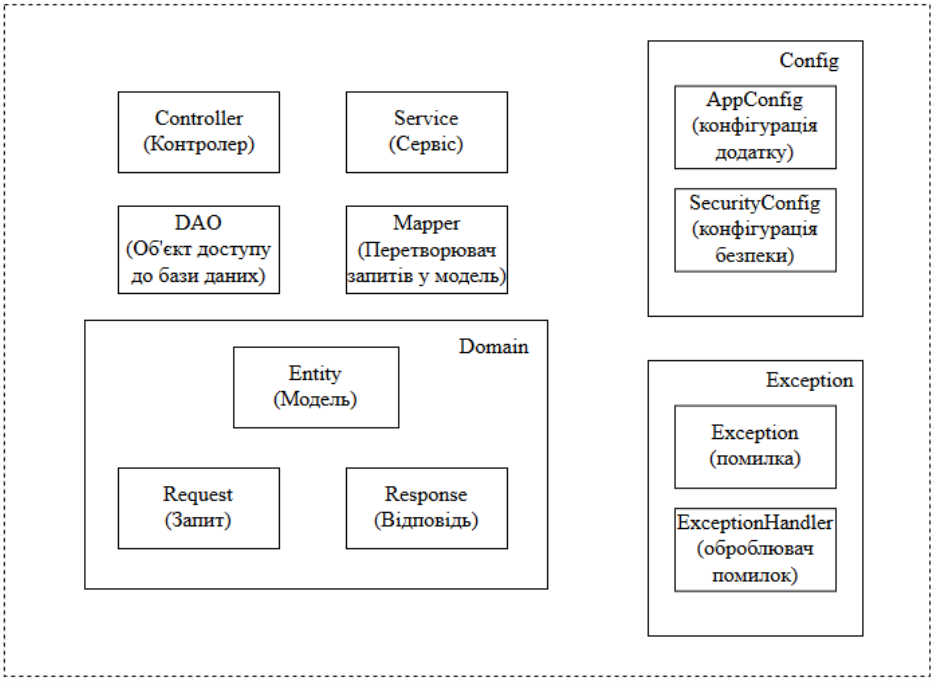

Рисунок 3.3 - Структурна схема REST веб-сервісу

4. АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ-СЕРВІСІВ ТА ОБГРУНТУВАННЯ ЇХ ВИКОРИСТАННЯ

4.1 Фреймворк Spring та Spring Boot

Spring Framework – універсальний фреймворк з відкритим кодом для Java-платформи. Широко використовується для розробки великих та реальних проектів. Spring є інструментом для вирішення типових питань, з якими працюють розробники та компанії, дозволяючи економити час та фінанси.

Spring являє собою фреймворк, який складається з наступних основних компонентів:

1. Spring IOC - контейнер для реалізації інверсії управління (Inversion of Control, IOC);
2. Spring Data - фреймворк для роботи з БД;
3. Spring Transaction - фреймворк управління транзакціями під час роботи з БД;
4. Spring MVC - реалізація шаблону MVC;
5. Spring Security - фреймворк для реалізації безпеки додатків;
6. Spring Testing - фреймворк для тестування додатків, розроблених за допомогою Spring.

Spring IOC

Інверсія управління являє собою принцип, при якому компоненти програми не створюють собі поведінку, а отримають її ззовні. Під «ззовні» зазвичай мається на увазі контейнер, який містить в собі об’єкти визначених компонентів програми та має змогу надавати їх іншим компонентам. Такий підхід називають впровадженням залежностей (англ. Dependency Injection). Наприклад, якщо клас А використовує клас В, то він повинен не створювати його всередині себе, а отримати об’єкт ззовні, тобто з контейнеру. Такий

підхід дозволяє винести створення об’єктів компонентів програми в загальну конфігурацію, яку простіше змінювати та розширювати за потребою.

Spring Data

Spring у своєму складі має фреймворк для роботи з БД – Spring Data - який дозволяє працювати з різними типами БД найбільш зручним способом. Фреймворк надає розробнику зручний інтерфейс та конфігурацію, яку необхідно визначити для роботи з конкретною БД та ізолює від необхідності створювати підключення вручну, писати запити в БД.

Зручність Spring Data полягає у тому, що розробнику достатньо написати назву методу за спеціальним синтаксисом, який фреймворк зрозуміє та за визначенням якого зробить запит до бази даних. Розробнику не потрібно писати реалізацію цього методу - він буде реалізований самим фреймворком.

Також фреймворк роботи з БД дозволяє конфігурувати підключення бази даних. Для підключення до бази даних розробнику необхідно задати чотири обов’язкових параметри: URL підключення до БД, ім’я користувача, пароль та назву драйверу, що використовується для підключення до БД. Всі ці параметри задаються в конфігураційному файлі проекту, та не потребують перекомпіляції проекту в разі їх зміни.

Spring Transaction

Транзакцією називають групу послідовних операцій з БД, що є логічною одиницею роботи з нею. При роботі з БД для збереження цілісності інформації необхідно представляти сукупність запитів в БД як одне ціле. Лише при успішному виконанні сукупності запитів можна зберегти результат - інакше БД може залишитися в неконсистентному стані та є ризик порушити цілісність даних БД, унеможливити роботу з нею. Транзакції використовуються для забезпечення надійних робочих елементів, які дозволяють правильно відновити роботу у випадку програмних чи апаратних збоїв та зберегти цілісність існуючих даних.

Одною з найбільш розповсюджених вимог до транзакцій є набір ACID (Atomicity, Consistency, Isolation, Durability):

1. Atomicity - атомарність - гарантує, що ніяка транзакція не може бути зафіксована в системі частково;
2. Consistency - узгодженість - визначає стабільний стан системи до початку транзакції та після її виконання;
3. Isolation - ізольованість - гарантує, що при виконанні паралельних транзакцій одночасно одна транзакція не впливає на результат виконання іншої;
4. Durability - надійність - гарантує, незалежно апаратних проблем система залишиться в консистентному стані.

Всі визначені компоненти є критичними при роботі з БД та забезпечують цілісність даних. Відповідно до вимог існують рівні ізоляції для забезпечення цих вимог. Spring Transaction надає простий на ефективний метод налаштування транзакцій.

Spring MVC

Фреймворк Spring MVC являє собою реалізацію архітектурного шаблону MVC, надаючи зручний інтерфейс для розробників. Фреймворк надає можливість розроблювати незалежні один від одного компоненти шаблону MVC (Модель, Вигляд, Контролер).

Spring MVC використовує в своїй основі компонент DispatcherServlet, що відповідальний за обробку всіх HTTP-запитів та HTTP-відповідей. Обробка HTTP-запиту в Spring MVC показана на рисунку 4.1.

Розглянемо послідовність подій, які відбуваються при надходженні вхідного HTTP-запиту до DispatcherServlet:

1. Після отримання HTTP-запиту DispatcherServlet використовує HandlerMapping для визначення відповідного контролеру, який повинен бути викликаний;

2. Контролер приймає запити відповідно до URL та типу запиту, делегує виконання необхідної логіки на інші компоненти та повертає ім'я Вигляду в DispatcherServlet.
3. ViewResolver визначає Вигляд на основі відповіді контролера.
4. Після того, як *Вигляд* знайдено, DispatcherServlet підставляє дані з *Моделі* у *Вигляд*, який в результаті буде відображено в браузері.

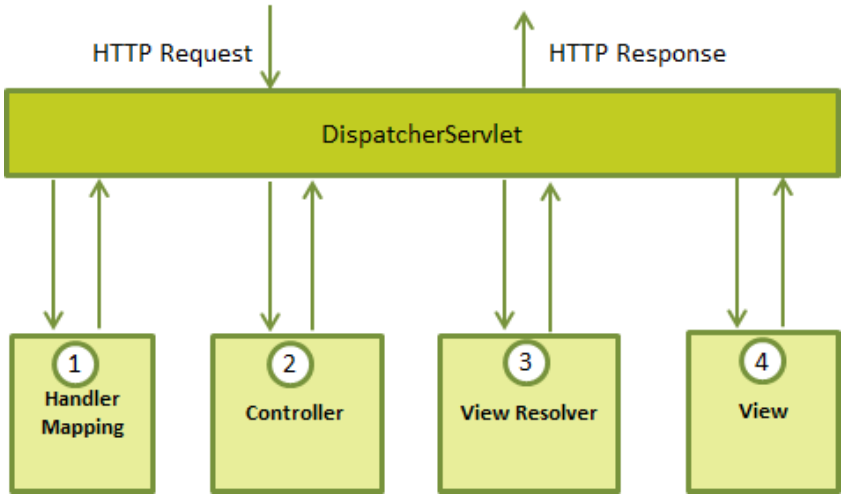

Рисунок 4.1 - Обробка запиту робочого процесу в Spring MVC

Spring Security

Spring має інструменти для забезпечення безпеки додатків – Spring Security. Даний фреймворк має зручний інтерфейс для налаштування безпеки: налаштування CORS, необхідності вводу логіну-паролю, забезпечення безпеки за допомогою використання спеціальних токенів тощо. Більшість найбільш використаних налаштувань можуть задаватися в файлах конфігурації проекту, за потреби гнучкої конфігурації – безпосередньо в коді додатку

Spring Testing

Для тестування використовується набір інструментів, що дозволяють зручним та ефективним чином тестувати додатки, побудовані на Spring MVC, тестувати роботу з БД, прикладні програми тощо.

Spring Boot

Spring Boot являє собою фреймворк для спрощення налаштування додатків, побудованих на фреймворку Spring. Велика кількість компонентів, що входить до фреймворку Spring, робить незручним налаштування взаємодії між ними, підтримку версій, написання конфігурації тощо. Spring Boot дозволяє автоматизувати деякі моменти, що робить простими інтеграцію компонентів та їх налаштування.

Фреймворк Spring Boot є дуже зручним інструментарієм для вирішення більшості задач та часто використовується при реалізації веб-сервісів та прикладних програм. В даній дипломній роботі фреймворк використовується в якості головного інструментарію для реалізації генератора REST веб-сервісів.

При реалізації генератора фреймворк Spring Boot використовується для:

1. Спрощення програмного коду;
2. Зменшення об’єму програмного коду;
3. Автоматизації та спрощення шаблонних питань;
4. Зручності конфігурації генератора;
5. Простоти розширення генератора;
6. Можливості розширення функціоналу генератора;
7. Тестування.

REST веб-сервіс, що генерується, в своєї основі також має фреймворк Spring Boot. Це дозволяє отримати всі вищеперераховані переваги, а також, головне, зменшити кількість генеруючого коду та спростити його конфігурацію. Важливим аспектом використання даного фреймворку є його широке використання для розробки додатків на мові Java, що є зручним для розробників.

Єдиним недоліком використання даного фреймворку є його «ваговитість». До його складу входить багато компонентів, які часто потрібно підключати навіть при використанні лише малої частини їх функціоналу. В той же час, Spring Boot є добре оптимізованою платформою та «під капотом»

використовує ефективні рішення та алгоритми, що дозволяє зменшити вплив фреймворку на швидкодію додатків під час їх роботи.

4.2 Фреймворк Hibernate

Hibernate – фреймворк для Java-платформи, реалізація специфікації JPA для рішення так званого об’єктно-реляційного відображення (англ. Object-relation mapping, скорочено ORM).

Мета даного фреймворку досить проста – дати можливість представляти сутності таблиці БД у вигляді Java-об’єктів, прибираючи необхідність розробників працювати з SQL-запитами та таблицями напряму.

Проблема полягає в написанні і виконанні SQL-запитів, а також перетворенні записів з SQL-таблиць в Java-об’єкти. Це вимагає складної конфігурації, великої кількості додаткового функціоналу, який в більшій мірі може бути автоматизований за допомогою Hibernate.

ORM

Об’єктно-реляційне відображення – технологія програмування, яка зв’язує БД з концепціями об’єктно-орієнтованих мов програмування. Останнє може бути інтерпретоване звичайним Java-об’єктом, який несе собі структуру запису з БД.

JPA

JPA (англ. Java Persistence API) – специфікація для Java ORM фреймворків, що визначає загальний інтерфейс та принципи їх роботи. JPA являє собою простий опис функціювання ORM фреймворків.

Hibernate являє собою безпосередню реалізацію JPA.

Переваги Hibernate:

- фреймворк дозволяє працювати з SQL-таблицями за допомогою Java-коду;

- дозволяє позбутися явного використання SQL-запитів в конфігурації та програмному коді;
- виконує перетворення записів з таблиць БД в Java-об’єкти;
- значно зменшує кількість програмного коду;
- полегшує конфігурацію;
- дозволяє задавати структуру БД та її таблиць безпосередньо з Java-коду;
- дозволяє працювати з БД абстрагуючись від її типу та діалекту SQL-запитів;
- інтегрований з фреймворком Spring Boot.

Hibernate має складну реалізацію, через яку за історію своєї еволюції фреймворк мав деякі недоліки в плані неочікуваної поведінки та швидкодії, але з останніми версіями ці недоліки повністю (або майже повністю) були усунені.

Генератор REST веб-сервісів не працює з БД, тому для його реалізації Hibernate не використовується. Фреймворк використовується для генеруючого веб-сервісу, який працює з БД за допомогою Hibernate. Фреймворк доцільно використовувати завдяки переліченим вище перевагам. Головним є те, що він дозволяє зменшити кількість генеруючого коду та полегшити інтеграцію з багатьма типами БД, а також їх конфігурацію.

В той же час, при генеруванні веб-сервісу, в основі якого лежить Hibernate викликає деякі труднощі перетворення вхідних від користувача SQL-запитів створення таблиць на Java-код, з якого безпосередньо за допомогою Hibernate задається структура таблиць БД та операції роботи з ними. Задача перетворення є однією із головних для даної дипломної роботи.

Прикладом Hibernate-сутності може слугувати наступний Java-клас:

```java
@Entity
@Table(name = "car")
public class Car {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        @Column(name = "car_id")
        private Long id;

        @Column
        private String brand;

}
```

В такому Java-класі визначається:

- назва таблиці;
- первинний ключ таблиці, стратегію його генерування;
- назви та параметри стовбців в таблиці;
- зв’язки з іншими таблицями тощо.

5. РОЗРОБКА ГЕНАРАТОРА ВЕБ-СЕРВІСІВ

5.1 Генерування базового проекту веб-сервісу

Java проектом прийнято називати сукупність Java-класів та пакетів, що виконують перелік взаємопов'язаних дій та розцінюються як одне ціле. Декілька Java-класів, що мають спільну ціль, доцільно об'єднувати в окремі каталоги, які називають пакетами. Проект має певну структуру, тобто визначений порядок розташування пакетів та їх рівні, вкладеність один в іншого. Існують визначені домовленості щодо структури проекту, що роблять структуру проекту зрозумілою та інформативною. Стандартизована структура дозволяє сучасним IDE автоматично налаштовувати проект: шляхи до файлів ресурсів, можливість автоматичного запуску тестів тощо. Базовий Java-проект має структуру, зображену на рисунку 5.1.

На рисунку кореневий пакет носить назву проекту artifactId. Під назвою підприємства groupId мається на увазі домен організації чи підприємства, наприклад tk.roydgar. В корені цього пакету можуть розміщуватися довільні пакети та Java-класи, що визначає розробник.

В пакеті main містяться основний виконуваний код Java-програми.

В пакеті resources знаходяться зовнішні ресурси програми: файли конфігурації, статичні ресурси, зображення, текстові файли тощо. Пакет ресурсів визначається автоматично IDE, або може бути заданий вручну. Після задання файлу ресурсів Java JRE не потребує вказання шляху до пакету ресурсів відносно кореню проекту. Достатньо вказати назву ресурсу без абсолютних або відносних шляхів.

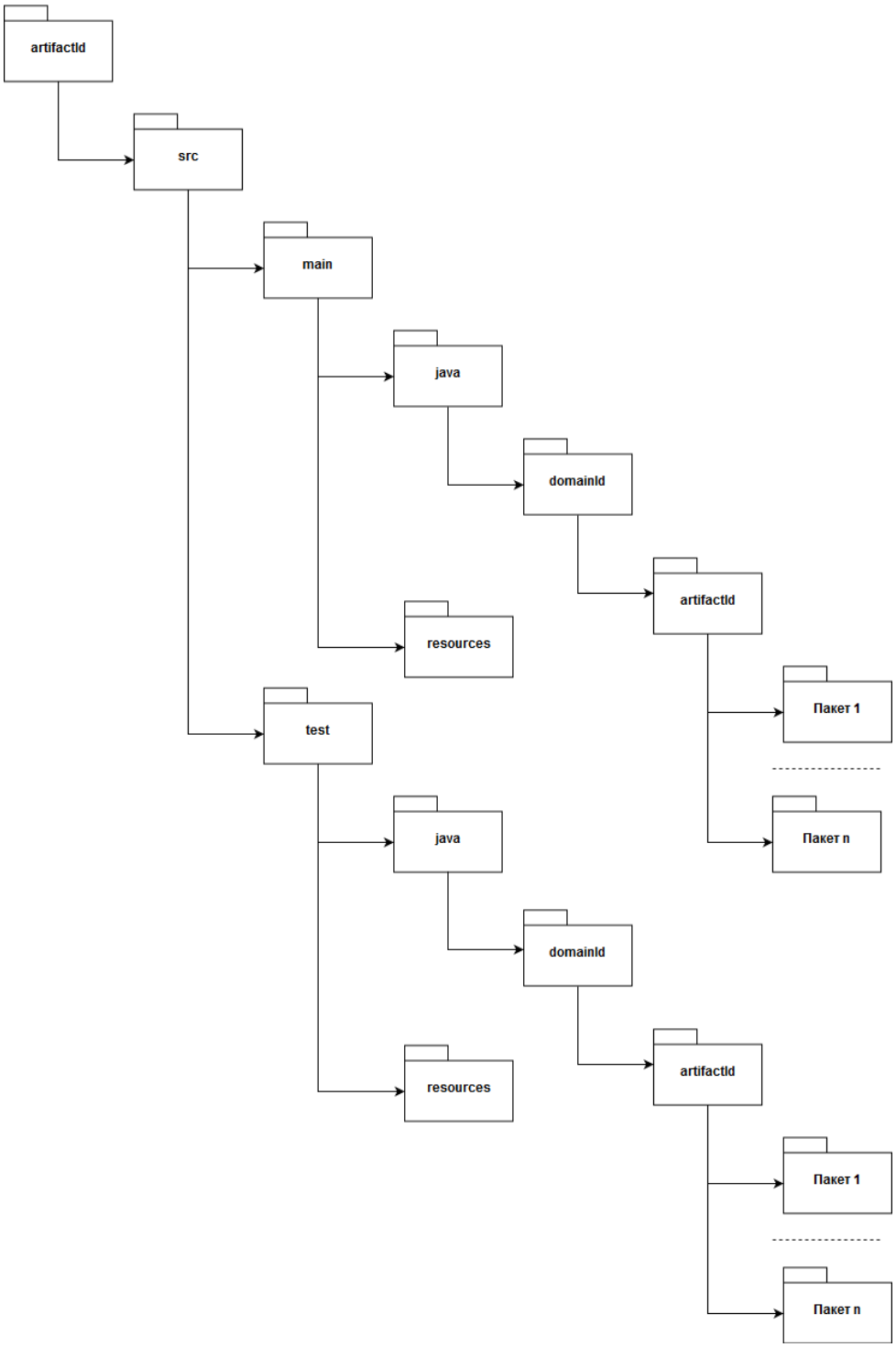

Рисунок 5.1 - Базова структура Java-проекту

В пакеті test знаходяться Java-класи тестів. Інша ієрархія пакетів аналогічна ієрархії в пакеті main. IDE ідентифікує пакет тестів автоматично, за потреби його можна задати вручну. Завдяки визначенню пакету тестів,

система автоматичної збірки проектів дозволить запускати тести автоматично, або вручну за допомогою однієї команди.

Для генерування базового Java-проекту важливе визначення основних його залежностей. Під залежностями маються на увазі всі необхідні бібліотеки, фреймворки, які будуть використовуватися в проекті. Для функціювання генеруючого проекту необхідні наступні залежності:

1. Spring Web (Spring MVC) для реалізації архітектурного шаблону MVC;
2. Spring Data для роботи з БД. До складу Spring Data також входить фреймворк Hibernate;
3. Lombok для генерування шаблонного коду гетерів, сетерів та конструкторів в Java-класах.

Назви залежностей, а також версія фреймворку Spring Boot (та відповідно всіх його компонентів) задаються в конфігураційному файлі під назвою application-springinitializr.yml:

application-springinitializr.yml

```
spring-initializr:
  springBootVersion: 2.1.5.RELEASE
  dependencies:
    - data-jpa
    - web
    - lombok
  sqlDialectToDependencyMap:
    MY_SQL: mysql
```

Рисунок 5.2 - Конфігураційний файл основних залежностей проекту

Для роботи генеруючого проекту з певним типом БД необхідно визначити конфігурацію в вищенаведеному файлі. Так, в параметрі під назвою sqlDialectToDependencyMap задаються назви залежностей в залежності від типу БД, визначеної користувачем для генеруючого веб-сервісу.

Для генерування базового проекту доцільно використовувати існуючий веб-сервіс для ініціалізації Spring Boot проекту під назвою SpringInitializr. Веб-сервіс відповідальний за генерування базової структури проекту, генерування файлу автоматичної збірки проекту з вказаними залежностями.

Для використання веб-сервісу SpringInitializr розроблений компонент під назвою SpringInitializrClient, що відповідальний за звернення до останнього шляхом GET HTTP-запиту. Запит містить наступні параметри:

- groupId – домен організації чи підприємства;
- artifactId – назва генеруючого проекту;
- type – тип системи автоматичної збірки (Maven або Gradle);
- language – мова програмування;
- bootVersion - Версія фреймворку Spring Boot;
- packaging – пакування проекту при збірці (War або Jar);
- javaVersion – версія мови Java;
- description – короткий опис проекту;
- style – назва залежності. Можливий перелік довільної кількості параметрів style з назвами залежностей.

Назви залежностей та версія фреймворку Spring Boot визначаються в файлі конфігурації, що наведено на рисунку 5.2. Параметр Language є статичним та має значення «java». Решта параметрів визначаються користувачем за допомогою користувацького інтерфейсу.

SpringInitializrClient приймає зібрані параметри та робить GET запит на веб-сервіс SpringInitializr. Результатом запиту є масив байтів, який представляє собою файл .zip, що містить згенерований базовий проект. Наприклад, URL ззапиту може мати вигляд:

https://start.spring.io/starter.zip?type=maven-project&language=java&bootVersion=2.1.5.RELEASE&baseDir=demo&groupId=com.example&artifactId=demo&name=demo&description=Demo+project+for+Spri

ng+Boot&packageName=com.example.demo&packaging=jar&javaVersion=1.8&inputSearch=&style=web&style=lombok&style=data-jpa&style=mysql

Як результат, маємо згенерований базовий проект, що тимчасово знаходиться в оперативній пам’яті під час роботи генератора у вигляді масиву байтів. Результуючий проект має необхідну структуру пакетів, а також всі вказані залежності та може бути розрахований в довільну директорію, вказану користувачем.

5.2 Обробка SQL-запитів на створення таблиць в БД

Наступним кроком з генерування REST веб-сервісу є обробка SQL-запитів на створення таблиць в БД, які вводить користувач для задання структури БД.

Важливим питанням, що необхідно вирішити, є перетворення SQL-запитів на Java-об’єкти, які визначають структуру БД безпосередньо в Java-коді за допомогою фреймворку Hibernate. Тобто, SQL-запити на створення таблиць не виконуються та використовуються за прямим своїм призначенням. З однієї сторони, такі перетворення несуть в собі деякі складнощі, але з іншої - Hibernate значно зменшує кількість генеруючого коду. Також, використання Hibernate необхідне, якщо згенерований проект націлені використовувати інші розробники для його розширення.

Для обробки SQL-запитів доцільно використовувати сторонню бібліотеку, яка працює з різними SQL-діалектами. Генератор використовує бібліотеку під назвою Aliababa Druid. Зазвичай, такі бібліотеки є гнучкими та універсальними, але на виході дають складний результат, який треба привести до більш простого для генерування Hibernate-сутностей у вигляді Java-класів.

Alibaba Druid реалізує шаблон Відвідувач (англ. Visitor). Шаблон дозволяє відділити певний алгоритм від компонентів, на яких цей алгоритм повинен бути виконаний. Так як Alibaba Druid бібліотека містить в собі багато SQL сутностей, що представлені у вигляді Java-об’єктів, на кожну таку

сутність необхідно створювати свій Відвідувач, який буде «частина за частиною» збирати інформацію про створювану SQL-таблицю та приводити її до простого вигляду, як зображено на рисунку 5.3. Визначення SQL-таблиці в такому вигляді робить можливим генерування Hibernate-сутностей.

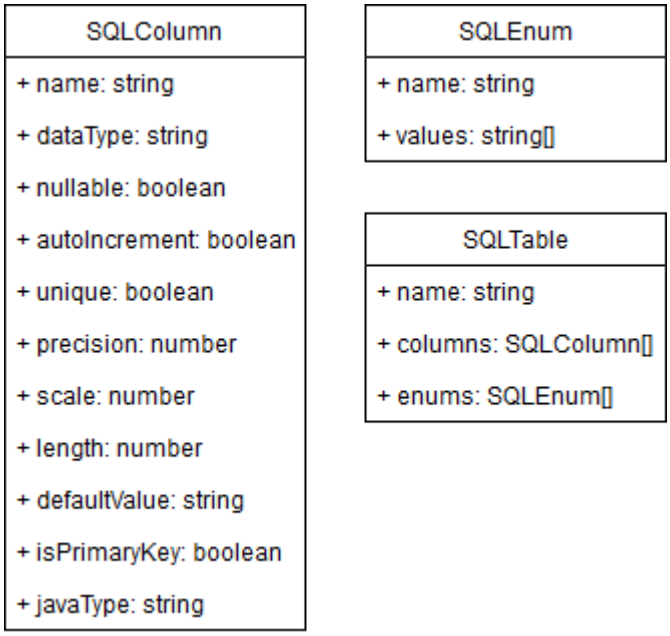

Рисунок 5.3 - Спрощене представлення SQL-таблиці у вигляді Java-об’єкту

Особливістю генератора є те, що він може приймати SQL-запити різних SQL-діалектів. Для підтримки такої можливості необхідно створити Відвідувачі, зазначені вище. За замовчуванням в коді генератора реалізована логіка щодо обробки SQL-виразів, що є спільними для всіх діалектів. Для обробки специфічних для діалекту виразів необхідно реалізувати конкретні Відвідувачі.

Важливим моментом є перетворення SQL-типів даних в Java-типи, так як фреймворк Hibernate оперує саме Java-типами даних. Для цього існує спеціальний файл конфігурації, який визначає ці перетворення. Наприклад, на рисунку 5.4 наведений конфігураційний файл для діалекту MySQL. Ці перетворення взяті з документації Hibernate та є оберненими, адже Hibernate виконує перетворення Java-типів даних в SQL-типи. Для конкретного SQL-діалекту необхідно задати свої параметри перетворень типів. Поле

typeClassification є службовим та визначає абстрактний клас типів, що необхідно для реалізації перетворень.

application-mysql.yml

```yaml
mysql:
  sqlTypeMap:
    bit:
      typeClassification: BOOLEAN
      javaType: BOOLEAN
    tinyint:
      typeClassification: NUMERIC
      javaType: INTEGER
    bool:
      typeClassification: BOOLEAN
      javaType: BOOLEAN
    boolean:
      typeClassification: BOOLEAN
      javaType: BOOLEAN
    smallint:
      typeClassification: NUMERIC
      javaType: INTEGER
    mediumint:
      typeClassification: NUMERIC
      javaType: INTEGER
    int:
      typeClassification: NUMERIC
      javaType: INTEGER
    integer:
      typeClassification: NUMERIC
      javaType: INTEGER
```

Рисунок 5.4 - Конфігураційний файл перетворень MySQL-типів даних у відповідні Java-типи

5.3 Визначення шляхів розташування генеруючих файлів проекту

Наступною не менш важливою задачею генератора є визначення шляхів, за якими будуть розташовуватися генеруючі Java-класи. На рисунку 5.1 зображена базова структура проекту та визначені кореневі пакети. Умовно Java-класи генеруючого проекту можна поділити на 2 типи: статичні та

динамічні. Такі позначення є умовними та мають місце лише при розробці даного дипломного проекту.

Статичні Java-класи мають одну реалізацію в проекті та не залежать від структури БД. Прикладом таких Java-класів є клас конфігурації, що генерується єдиним на весь проект.

Динамічні Java-класи можуть мати довільну кількість реалізацій в проекті та залежать від структури БД. Наприклад, для кожної таблиці БД необхідно створити свою Hibernate-сутність для роботи з нею.

Наведемо перелік та опис всіх Java-класів, які необхідно згенерувати.

Статичні Java-класи:

1. Config - configuration - Java-клас конфігурації проекту;
2. SecurityConfig - security configuration – Java-клас конфігурації безпеки;
3. NotFoundException - являє собою помилку відсутності запитуваного ресурсу;
4. ControllerAdvice - клас обробки помилок, що виникають під час роботи веб-сервісу.;
5. Properties - файл параметрів проекту, в якому можна задати порт, на якому запускається сервер, параметри підключення до БД тощо. Не є Java-класом, носить розширення .yml.

Динамічні Java-класи:

1. Controller - реалізація компоненту Контролер з шаблону MVC;
2. CreateRequest - клас запиту створення ресурсу;
3. UpdateRequest - клас запиту оновлення ресурсу;
4. DTO - клас відповіді серверу, що описує ресурс;
5. Entity - клас Hibernate-сутності, що задає структуру таблиці БД;
6. Enum - клас, що описує SQL-тип Enum;
7. Repository - клас реалізації DAO зі REST веб-сервісу, зображену на рисунку 3.3;

8. Service – клас реалізації Service, зображеного на рисунку 3.3;
9. RequestMapper – клас перетворювача запитів CreateRequest та UpdateRequest в клас Entity, визначених на рисунку 3.3;
10. ResponseMapper – клас перетворювача класу Entity в клас DTO, визначених на рисунку 3.3.

В коді генератора всі вищеперелічені класи визначені у вигляді констант, що визначають тип генеруючого Java-класу.

Так як кожен динамічний клас створюється для кожного визначення таблиці в БД, їх доцільно генерувати в спільний пакет. Наприклад, Service Java-класи можна генерувати в пакет під назвою service. Такий підхід дозволяє згрупувати компоненти програми, зробити її структуру простою та зрозумілою.

Для визначення шляхів розроблено Java-клас під назвою FileNameResolver. Мета компоненту полягає у визначені шляху, за яким будуть розташовуватися генеруючі Java-класи відносно їх типу, наведеного вище. Також компонент відповідальний за визначення імен генеруючих файлів. Для реалізації вказаних дій компонент використовує параметри проекту, вказані в конфігураційному файлі генератора. В конфігураційному файлі генератора можна визначити назву пакетів, в яких будуть розташовуватися генеруючі файли, формат їх назви та розширення.

Детальний опис конфігураційних файлів та структури генеруючого проекту знаходяться в додатках

5.4 Підготовка контексту для генерування файлів проекту

Контекстом для генерування є визначення SQL-таблиць та всі необхідні параметри від користувача, а також службові дані для визначення всіх необхідних імпортів, назв Java-класів тощо. Даний контекст використовується під час генерування Java-файлів, визначаючи динамічний вміст генеруючих

файлів. Для визначення контексту перед кожним генеруванням ефективним підходом є написання програмних правил, які будуть мати спільний інтерфейс. Кожне правило є відповідальним за визначення значень для контексту, що пов’язані спільною тематикою. За необхідністю простою є реалізація нових правил, а також модифікація існуючих. Для підготовки контексту генератор має два основних інтерфейсу компонентів.

TemplateContentProviderRule відповідає за підготовку контексту, приймаючи як параметр параметри генерування проекту та визначення SQL-таблиці. Правила, що реалізують даний інтерфейс, викликаються кожен раз для кожного визначення SQL-таблиці та використовуються для генерування Java-класів, що залежать від визначення SQL-таблиці

TemplateStaticContentProviderRule відповідає за підготовку контексту, приймаючи як параметр конфігурацію генерування проекту. Правила, що реалізують даний інтерфейс, використовуються для генерування Java-класів, що не залежать від визначення SQL-таблиці.

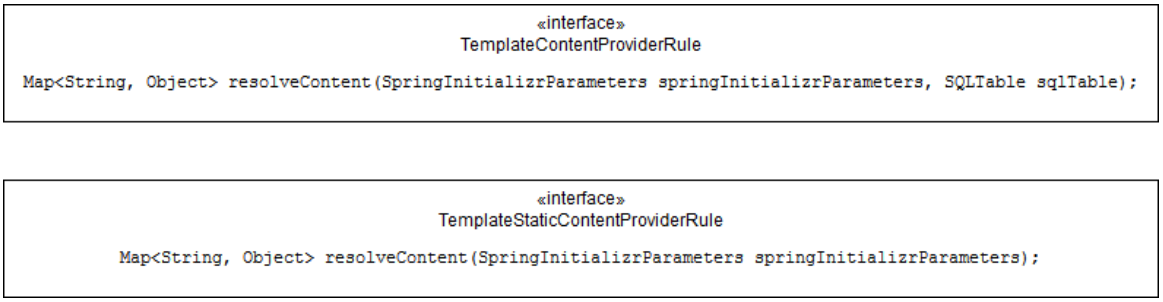

Рисунок 5.5 - Визначення інтерфейсів правил підготовки контексту для генерування

Кожне правило, що реалізує дані інтерфейси, використовує параметри для підготовки контексту та вертає його як структуру даних Map. Кожен елемент в Map зберігаються у вигляді «ключ-значення», де ключем є назва змінної що буде використовуватися в Template Engine шаблонах генератора, а значенням – безпосереднє значення змінної, на яке в шаблоні замінюється її назва.

Розроблені наступні реалізації TemplateContentProviderRule:

1. ClassNameTemplateContentProviderRule - використовується для визначення назв всіх генеруючих Java-класів;
2. ImportTemplateContentProviderRule - необхідний для визначення імпортів одних генеруючих Java-класів в інші;
3. PackageTemplateContentProviderRule - використовується для визначення назв пакетів, які прописуються в генеруючих Java-класах;
4. SQLTableTemplateContentProviderRule - постачає в контекст визначення SQL-таблиці.

Реалізації TemplateStaticContentProviderRule:

1. ClassNameStaticTemplateContentProviderRule - використовується для визначення назв всіх генеруючих Java-класів;
2. ImportStaticTemplateContentProviderRule - необхідний для визначення імпортів одних генеруючих Java-класів в інші;
3. PackageStaticTemplateContentProviderRule - використовується для визначення назв пакетів, які прописуються в генеруючих Java-класах.

5.5 Генерування файлів проекту

Генерування проекту реалізується за допомогою так званого Template Engine (движок шаблонів), який використовується для генерування текстових даних або файлів за заданим шаблоном, динамічно підставляючи дані з підготовленого раніше контексту.

Шаблоном для генерування є текстовий файл або рядок, що містить статичні дані та назви змінних, які замінюються значеннями з контексту. Кожен Template Engine має свій синтаксис вказання змінних в шаблоні, а також підтримує базові оператори розгалуження та цикли.

За наявності підготовленого контексту, що містить всі необхідні для генерування значення, можливе генерування файлів проекту. Для генерування використовується бібліотека Velocity Template Engine. Головним компонентом бібліотеки є Java-клас під назвою VelocityEngine, що має ряд методів для обробки шаблонів. Один із таких методів приймає шлях до файлу шаблону, що знаходиться в ресурсах генератора та підготовлений контекст, який використовується для обробки шаблонів. Результат роботи VelocityEngine може бути збережений в оперативній пам'яті у вигляді строки, масиву байтів, або записаний у зовнішній файл.

Для кожного генеруючого типу файлу розроблений свій шаблон, який визначає його результуючий вміст. Файл шаблону знаходиться в ресурсах проекту та має розширення «.vm». Наприклад, для генерування Java-класу типу Enum реалізовано наступний шаблон enum.vm:

```
package $enum-package;

public enum $enum-class {
        #foreach($enum-value in $enum-values)
                $enum-value,
        #end
}
```

Так, синтаксис $var визначає назву змінної, значення якої підставляється під час обробки шаблону. Управляюча конструкція #foreach являє собою цикл, що проходиться по заданому масиву даних. Як зазначалося вище, всі змінні, що використовуються в шаблоні, необхідні бути визначені в контексті, щоб движок мав змогу замінити назви змінних на їх значення.

Для компонування всіх компонентів програми та запуску процесу генерування розроблений компонент під назвою Generator:

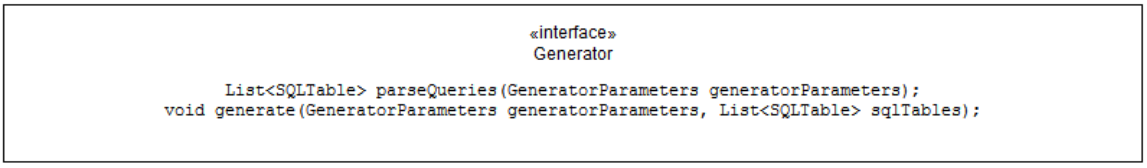

Рисунок 5.6 - Визначення інтерфейсів виконання задач систем автоматичної збірки проектів

Метод parseQueries() відповідальний за обробку SQL-запитів на створення таблиць, введених користувачем, на перетворення їх в представлення SQLTable. Як параметр, метод приймає параметри, що ввів користувач в користувацькому інтерфейсі.

Метод generate() також приймає параметри, введені користувачем, підготовлює контекст, генерує файли проекту за шаблонами для кожного визначення SQL-таблиці та генерує проект в директорію, вказану користувачем. Компонент Generator не виконує перелічені дії вручну, а делегує їх виконання іншим компонентам системи.

5.6 Збірка та запуск згенерованого веб-сервісу

Для збірки та запуску згенерованого веб-сервісу використовуються наведені раніше інструменти автоматичної збірки проектів. Для цього був розроблений компонент під назвою AutomationBuildGoalExecutor:

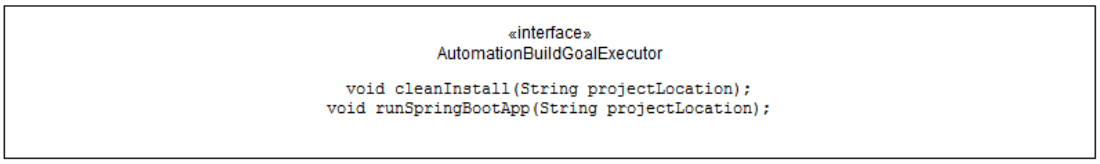

Рисунок 5.7 - Визначення інтерфейсів виконання задач систем автоматичної збірки проектів

Для збірки проектів використовується метод cleanInstall(), для запуску – runSpringBootApp(). Обидва методи приймають шлях до кореню згенерованого проекту, який визначається користувачем в користувацькому інтерфейсі.

Генератор підтримує генерування проекту, який мати в своєї основі одну з двох систем автоматичної збірки проектів – Maven або Gradle. Відповідно, в генераторі розроблено дві реалізації інтерфейсу AutomationBuildGoalExecutor: MavenAutomationBuildGoalExecutor та

GradleAutomationBuildGoalExecutor, що відповідальні за виконання команд системи автоматичної збірки проектів.

Збірка та запуск проекту виконується за допомогою Maven та Gradle команд. Для цього необхідно за допомогою консольних команд перейти в корінь згенерованого проекту та виконати наступні команди:

Для Maven:

1. cleanInstall - «mvn clean install»;
2. runSrpingBootApp – «mvn Spring-boot:run».

Для Gradle:

1. cleanInstall - «./gradlew clean build»;
2. runSrpingBootApp – «./gradlew bootRun».

5.7 Розробка графічного інтерфейсу користувача

Для розробки графічного інтерфейсу використовується графічна бібліотека під назвою Swing. Архітектура Swing розроблена за принципом «Look and Feel», де «Look» визначає зовнішній вигляд компонентів, а «Feel» - їх поведінку.

Бібліотека має в своєму складі реалізації розташування компонентів у вікні, що дозволяють швидко та ефективно задати розміщення необхідних компонентів у ньому. До кожного компоненту можна прив’язати довільну подію, що буде виконуватися при натисканні на цей компонент, наведення на нього мишею тощо. До складу бібліотеки входять всі необхідні компоненти для розробки повноцінного інтерфейсу, наприклад, кнопки, списки, звичайний текст, компоненти для вибору файлу або шляху в файловій системі користувача тощо.

Метою розробки графічного інтерфейсу генератора є взаємодія з користувачем та збір параметрів для генерування. Графічний інтерфейс складається з чотирьох вікон:

1. Вікно вводу SQL-запитів на створення таблиці;
2. Вікно вводу параметрів проекту;
3. Вікно вводу параметрів підключення до БД;
4. Вікно вибору шляху збереження згенерованого проекту.

Вікно вводу SQL-запитів на створення таблиці приймає SQL-діалект, на якому користувач хоче писати SQL-запити на створення таблиць та самі SQL-запити. Якщо запитів декілька, вони можуть бути розділені за допомогою роздільника ‘;’. Також вікно вводу запитів має полосу прокрутки в разі, якщо запити займають багато рядків.

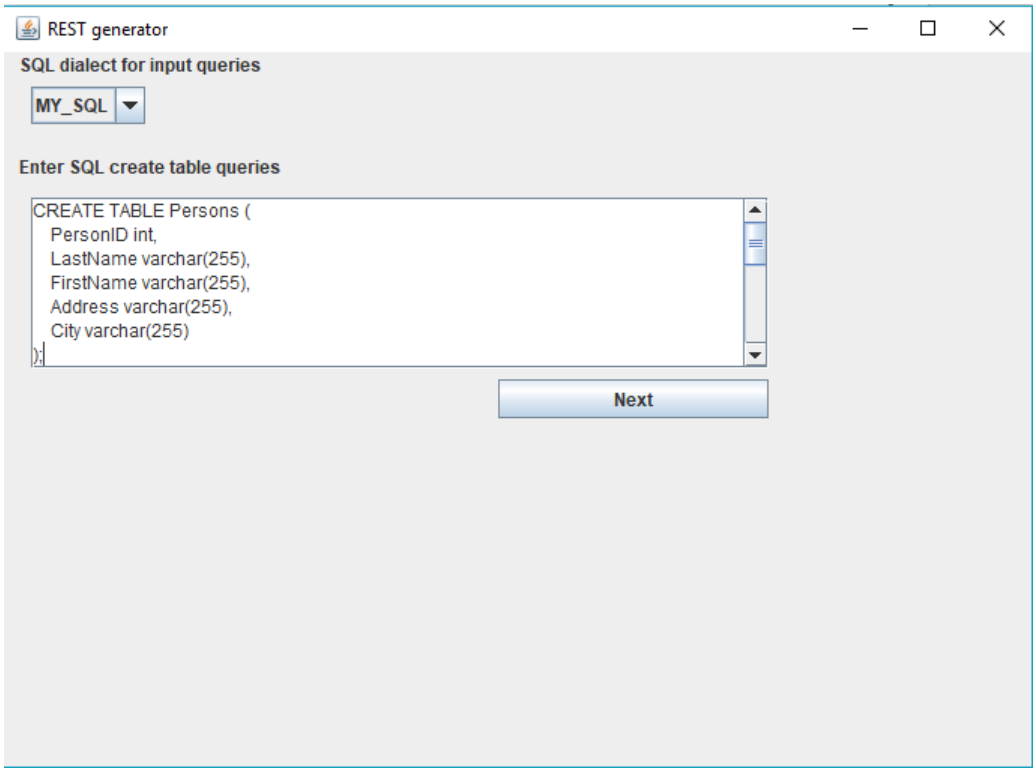

Рисунок 5.8 - Вікно вводу SQL-запитів на створення таблиць

За допомогою вікна вводу параметрів проекту користувач вводить наступні параметри:

1. Group id - домен організації чи підприємства;
2. Artifact id - назва проекту веб-сервісу;
3. Java version – версія мови програмування Java;
4. Description – короткий опис проекту;

5. Packaging - пакування проекту веб-сервісу: Jar (Java ARchive) чи War (Web ARchive);
6. Build system - система автоматичної збірки, за допомогою якого відбуватиметься збірка проекту;
7. Server port - порт, на якому запускається згенерований веб-сервіс. За замовчуванням має значення «8080».

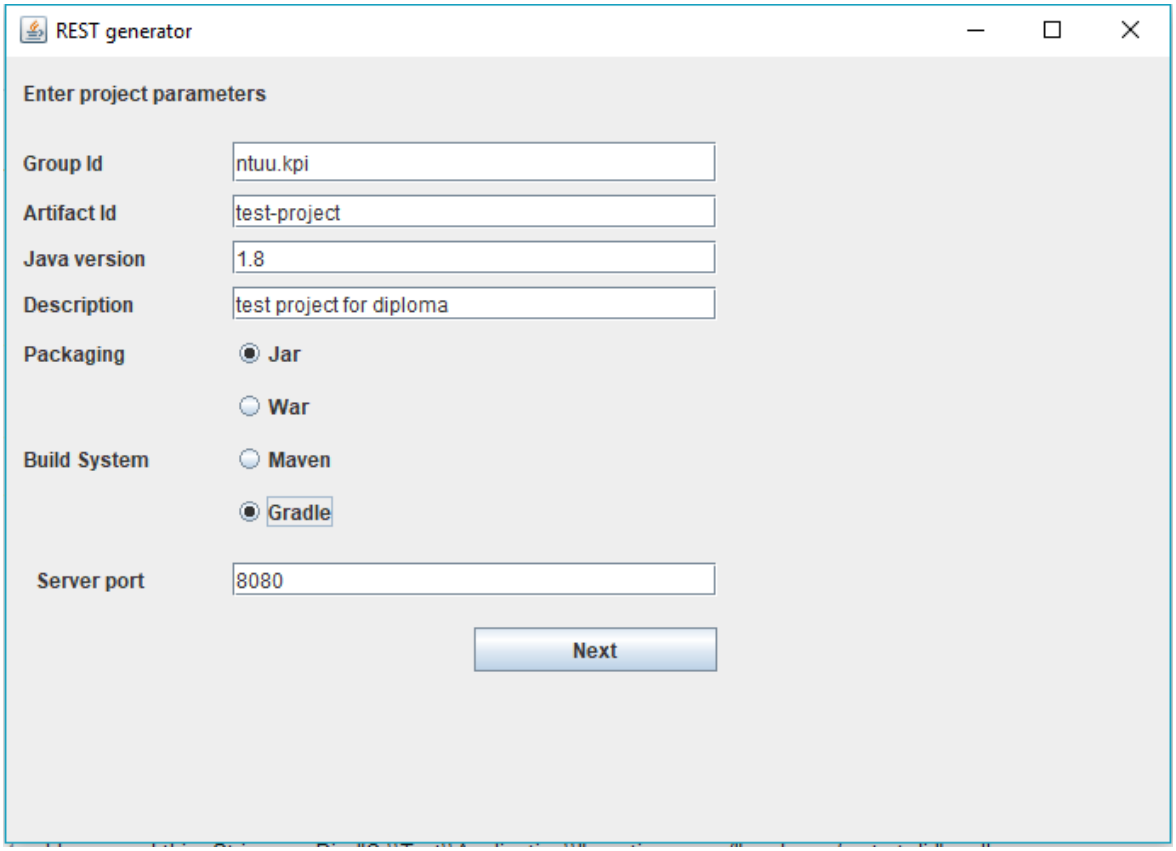

Рисунок 5.9 - Вікно вводу параметрів проекту веб-сервісу

Наступним вікном є вікно налаштування підключення до БД. Для цього користувачу необхідно ввести:

1. Datasource SQL dialect - діалект мови SQL;
2. URL - URL підключення до БД;
3. Username – ім’я користувача для доступу до БД;
4. Password – пароль користувача для доступу до БД;

5. Driver classname – назва драйверу підключення до БД. За замовчуванням визначається з визначеного раніше діалекту мови SQL.

Після введення перелічених параметрів користувач має змогу перевірити з'єднання з БД за допомогою кнопки «Test Connection» та впевнитись у правильності введених параметрів.

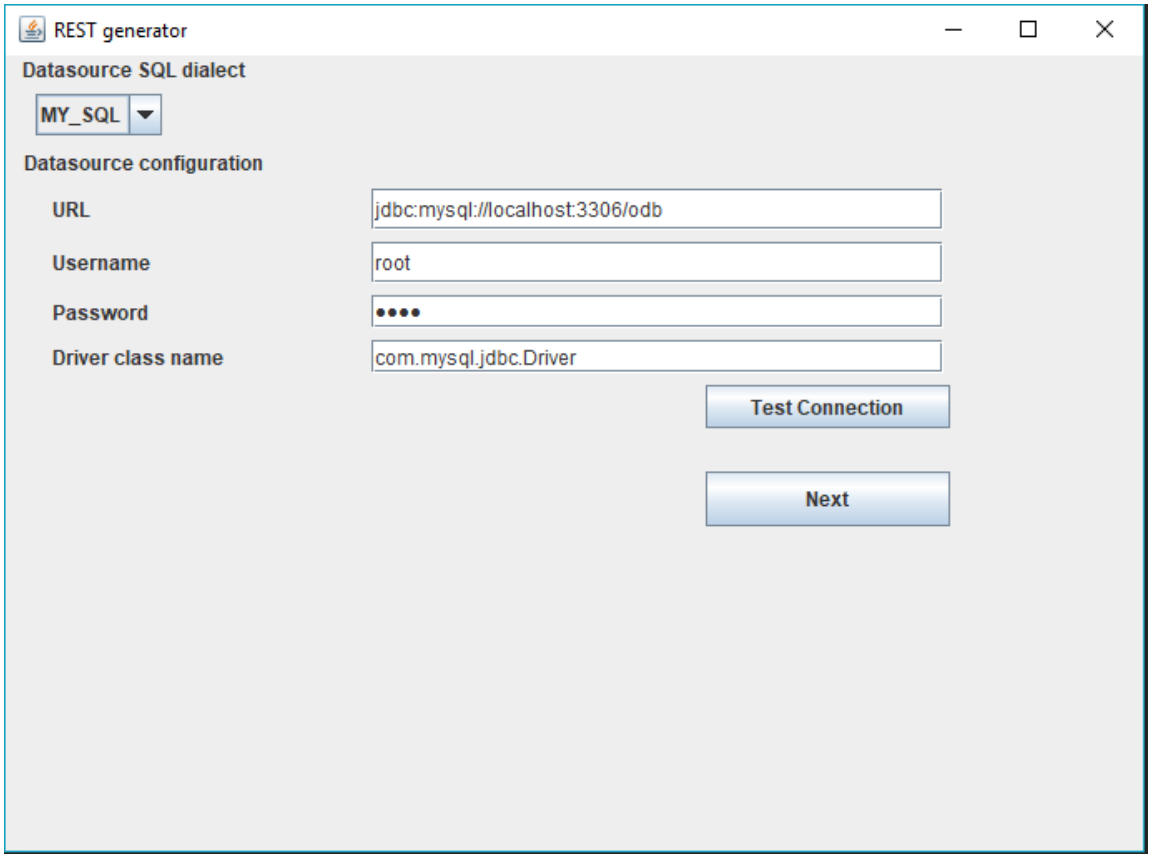

Рисунок 5.10 - Вікно вводу параметрів підключення до БД

У разі введення неправильних параметрів підключення при тестуванні з'єднання користувач отримує повідомлення про помилку з детальним описом причини.

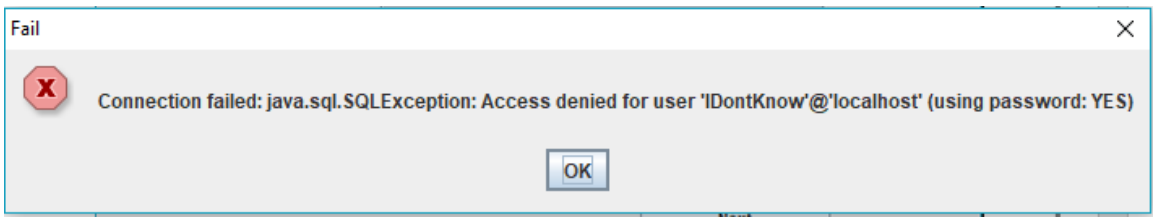

Рисунок 5.11 - Вікно повідомлення про помилку підключення до БД

У вікні вибору шляху збереження згенерованого проекту веб-сервісу користувач має змогу вибрати директорію в файловій системі куди буде збережений згенерований проект. За допомогою кнопки «Generate Project» проект генерується у вказану директорію. Також з останнього вікна користувач має змогу зібрати проект за допомогою кнопки «Clean and Build», а також запустити зібраний проект за допомогою кнопки «Run».

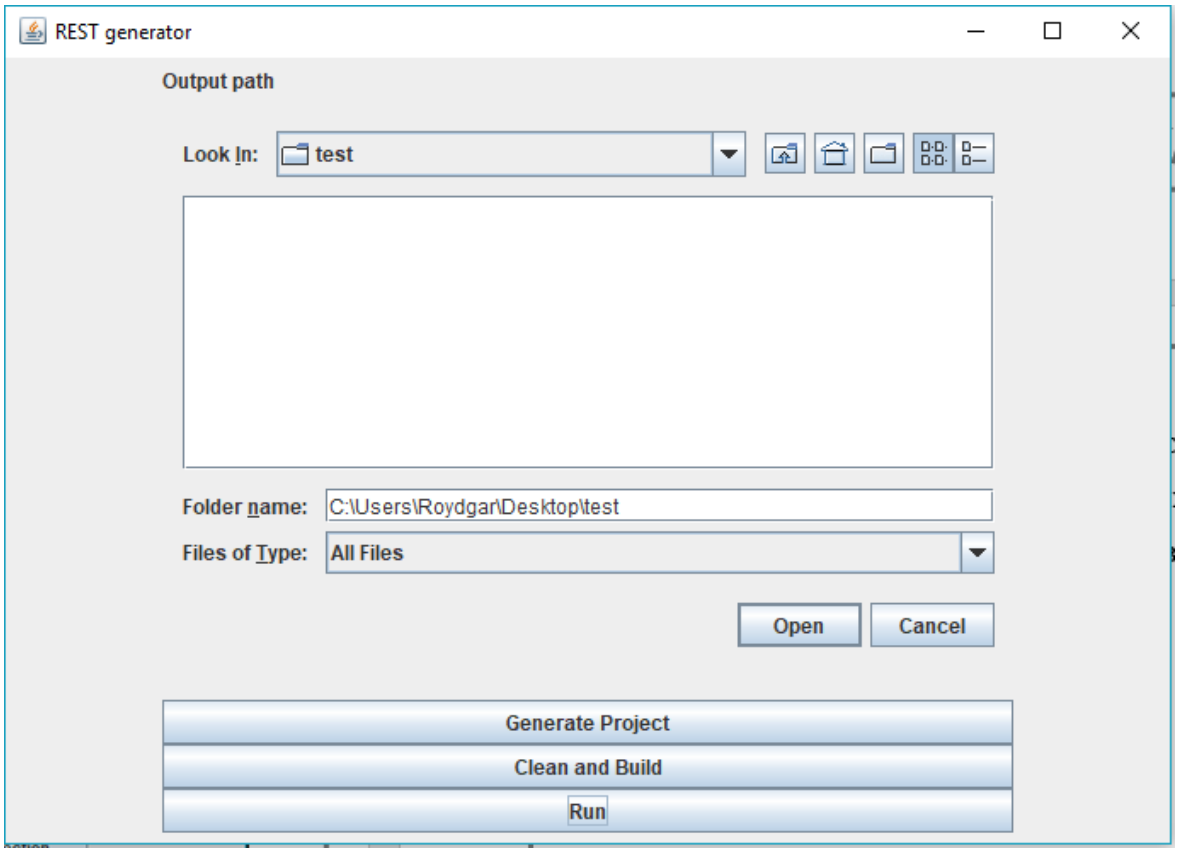

Рисунок 5.12 - Вікно генерування та запуску веб-сервісу

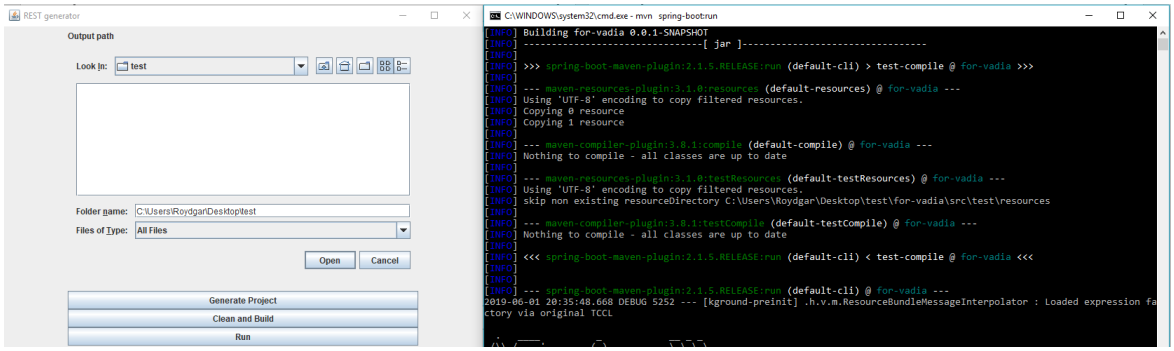

Рисунок 5.13 - Запущений командний рядок запуску проекту

Процес збірки та запуску згенерованого веб-сервісу відбувається в окремому вікні командного рядка.

Кожен рядок, що вводить користувач, перевіряється. Пусті рядки або занадто довгі не допускаються. Деякі рядки, такі як версія Java, перевіряються за допомогою регулярних виразів. Якщо користувач вводить неправильне значення - програма повідомляє про це в окремому вікні с відповідним повідомленням та не пропустить користувача на наступне вікно, доки він не введе коректні дані.

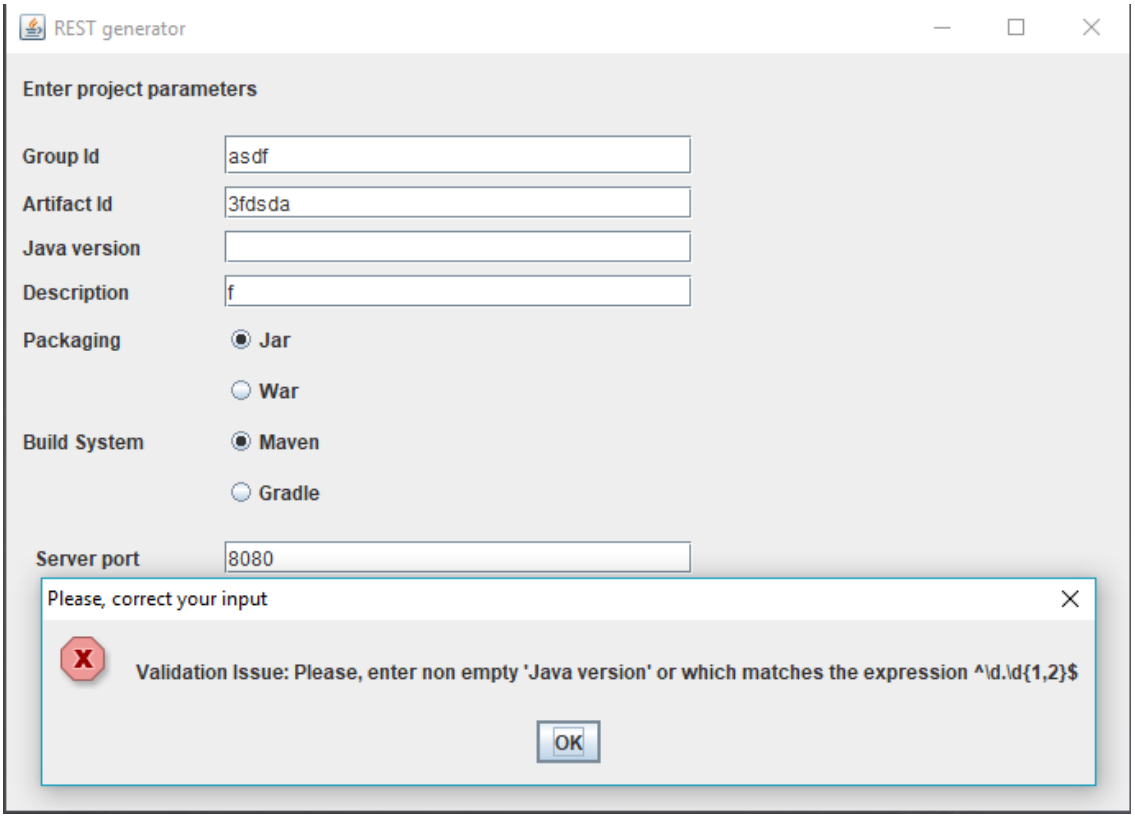

Рисунок 5.14 - Повідомлення про помилку вводу користувача

Формат повідомлень про помилку, регулярні вирази та параметри валідації, розмір вікон графічного інтерфейсу розміщуються у відповідному файлі конфігурації та можуть бути змінені без необхідності перекомпіляції генератора.

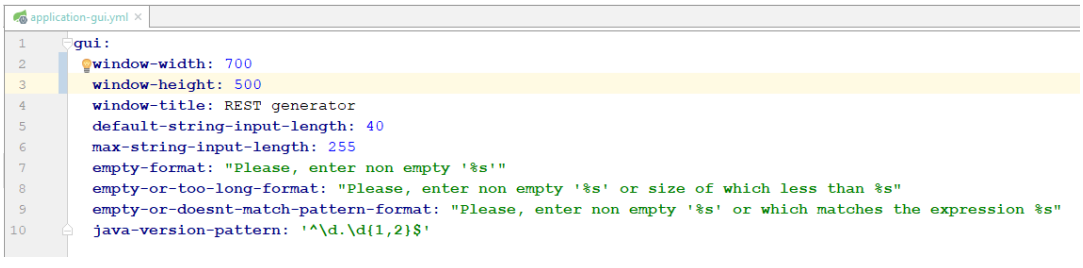

Рисунок 5.15 - Конфігураційний файл графічного інтерфейсу

5.8 Розробка REST-інтерфейсу генератора

Для використання функціоналу генератора в мережі поза межами одного лиш графічного інтерфейсу використовується REST-інтерфейс. Останній визначає URI запиту, зробивши запит на який у відповідь можна отримати масив байтів, що представляє собою файл з розширенням .zip. URI має вигляд «localhost:8080/api/generate». Зробивши POST-запит на даний ресурс, користувач або розробник може використовувати згенерований проект у вигляді .zip файлу. Розробка REST-інтерфейсу дозволяє користуватися генератором в мережі інтернет, роблячи запити на вказаний ресурс. Клієнтом може бути будь хто: веб-сторінка, мобільний додаток або додаток написаний на іншій мові програмування. Це досягається шляхом використання архітектурного рішення REST та обміну даними в форматі JSON.

POST-запит має містити в собі всі параметри, що були описані раніше. Щоб переглянути документацію, приклад запиту та параметри, що приймає REST-інтерфейс генератора, достатньо перейти за посиланням «localhost:8080/swagger-ui.html». При цьому відкриється веб-інтерфейс, що буде мати всю необхідну інформацію, а також з його допомогою можна зробити тестовий REST-запит.

Позначення «localhost:8080» є актуальним у разі запуску генератора на локальній машині. У випадку запуску генератора на сторонньому сервісі посилання та порт будуть інші.

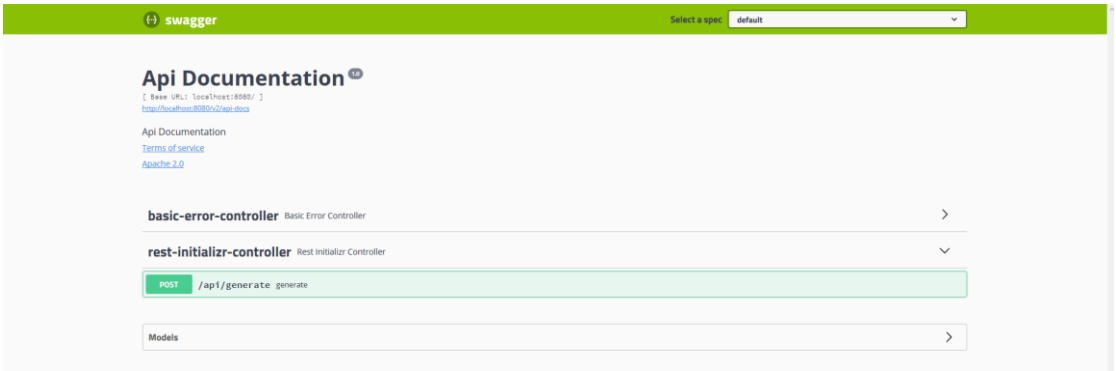

Рисунок 5.16 - Веб-інтерфейс генератора

5.9 Тестування функціоналу генератора

Для тестування генератора використовується фреймворк Spring Boot Test та бібліотека Mockito. Spring Boot Test дозволяє тестувати компоненти генератора загружаючи контекст додатку, з файлами конфігурації та всіма компонентами в контексті. Використання фреймворку необхідно, коли потрібно протестувати компонент, що має в своєму складі клас конфігурації, або залежить від інших компонентів. Для тестування простих незалежних класів достатньо бібліотеки Mockito.

Умовно тести генератора можна поділити на два типи: юніт-тести та інтеграційні тести. Юніт-тестування дозволяє перевірити на коректність окремі модулі коду програми, окремі методи та функції. Юніт-тестам надаються такі вимоги:

1. Тести повинні бути невеликими за розміром;
2. Кожен тест відповідальний за тестування тільки одного визначеного функціоналу;
3. Кожен тест має лише одне утвердження;
4. Тест не повинен залежати від іншого тесту;
5. Тести повинні бути читабельними та зрозумілими;
6. Тести повинні бути ізольовані від середовища (операційної системи, бази даних тощо.

Щоб зробити тести незалежними від середовища, вони повинні мати свій окремий контекст, незалежний від контексту основної програми, свою тестову БД та підключення до неї, окрему конфігурацію тощо. Також для ізолювання тестів від середовища використовуються так звані Mock-об’єкти, що являють собою фіктивну надбудову над реально існуючим об’єктом та мають його інтерфейс. Різниця полягає в тому, що при тестуванні замість звернення до реального об’єкту робиться виклик до Mock-об’єкту, а фіктивний об’єкт має довільну поведінку, визначену в тесті. Mock-об’єкти

доцільно використовувати, коли компонент програми робить запит до БД. Замість виконання реального запиту доцільно визначити фіктивні дані на Mock-об’єкті доступу до БД, що останній буде повертати. Таким чином, досягається модульність тестів та незалежність функціоналу, що тестується, від іншого функціоналу.

Задача інтеграційних тестів полягає в перевірці взаємодії модулів програми між собою, а також інтеграція окремих модулів в загальну систему. При реалізації інтеграційних тестів не використовуються Mock-об’єкти, а робиться виклик реального функціоналу. Щоб запобігти змінення стану БД для тестів використовується окрема БД, що є аналогічною. Використання тестової БД дозволяє зберегти стан реальної бази даних.

При тестуванні генератора використовуються як юніт-тести, так і інтеграційні. При юніт-тестуванні перевіряються окремі компоненти програми, наприклад, компоненти визначення шляхів, за якими будуть розташовуватися генеруючі файли. Інтеграційним підходом тестується сукупність компонентів, наприклад, зчитування SQL-запитів від користувача або повноцінне генерування проекту.

Так, при юніт-тестуванні компоненту визначення шляхів в нього передаються всі необхідні тестові параметри, такі як назва проекту, домен підприємства. Після виконання коду, що тестується, результат перевіряється з очікуваним. Якщо результат не є коректним - тест припиняє виконання та виводить відповідне повідомлення. При тестуванні важливо перевіряти випадки, коли в компонент передаються некоректні дані. В такому випадку програмний код, що тестується, повинен видати відповідну та конструктивну помилку, тест повинен її «піймати» та перевіряти на відповідність тієї, що очікується.

При інтеграційному тестуванні компоненту, що відповідальний за зчитування SQL-запитів, передаються різні SQL-запити на створення таблиць. Компонент зчитує їх, делегуючи деяку логіку на інші компоненти. Результат,

що повертається, перевіряється на відповідність очікуваному. Також, при передачі некоректного SQL-запиту компонент повинен видати помилку про неможливість його зчитування, що і перевіряють інтеграційні тести.

При інтеграційному тестуванні компонентів, що відповідальні за генерування файлів проекту, йому передаються тестові параметри. В результаті виконання коду, що тестується, на виході маємо згенерований в визначену директорію проект. Задача тестування полягає у перевірці наявності необхідних директорій та файлів, їх назв та вмісту.

ВИСНОВКИ

В даному дипломному проекті розроблений генератор REST веб-сервісу. Основною метою розробки є зменшення часу написання однотипного коду та швидка розробка веб-сервісу, що працює з заданою структурою БД. Програмний засіб не потребує у користувача наявності технічних навичок розробки веб-сервісів на мові Java.

Особливістю програмної розробки є можливість гнучкого налаштування генератора за допомогою файлів конфігурації, можливість розширення існуючого функціоналу, генерації нових компонентів.

Структура БД може бути задана за допомогою SQL-запитів на створення таблиць різних діалектів. Параметри генеруючого проекту налаштовуються у відповідному вікні графічного інтерфейсу, деякі з них для зручності мають значення за замовченням. Користувач має змогу налаштувати та перевірити з’єднання з різними типами баз даних, вказати порт, на якому запускати сервер, згенерувати налаштований проект, скомпілювати та запустити його використовуючи лише графічний інтерфейс.

Кожне введення даних користувачем перевіряється на коректність, щоб запобігти використання генератором некоректних даних, а також виникнення неочікуваних помилок при роботі з програмою. Всі параметри мають коротке пояснення в графічному інтерфейсі при наведення на них мишею.

Як результат, функціонал генератора може використовуватися користувачами, що працюють з БД, розробниками - для генерації проекту, що слугує відправною точкою для реалізації власного ПЗ та автоматизації шаблонних процесів розробки ПЗ. В мережі генератор може використовуватися іншими додатками за допомогою REST-інтерфейсу генератора.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Многоуровневые системы клиент-сервер. – Електрон. дані (1 файл). – 2010-2014. – Режим доступу: http://www.osp.ru/nets/1997/06/142618/ (дата звернення 14.04.2016). – Назва з екрану.
2. Introduction to the Spring Framework. – Електрон. дані (1 файл). – 2015-2016. – Режим доступу: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html (дата звернення 18.04.2016). – Назва з екрану.
3. Spring - MVC Framework Tutorial. – Електрон. дані (1 файл). – 2015-2016. – Режим доступу: http://www.tutorialspoint.com/spring/spring_ web_mvc_framework.htm (дата звернення 18.04.2016). – Назва з екрану.
4. Компоненты сетевого приложения. Клиент-серверное взаимодействие и роли серверов. – Електрон. дані (1 файл). – 2012. – Режим доступу: http://www.4stud.info/networking/lecture5.html (дата звернення 16.04.2016). – Назва з екрану.
5. The DCI Architecture: A New Vision of Object-Oriented Programming. – Електрон. дані (1 файл). – 2014. – Режим доступу: http://www.artima.com/articles/dci_vision.html (дата звернення 18.04.2016). – Назва з екрану.
6. Walls C. Spring in action / C. Walls.- Manning Publications., 2014. – 624p.
7. Eckel B. Thinking In Java / B. Eckel.- Prentice Hall., 2006 – 1150p.
8. Gutierrez F. Pro Spring Boot 2 / – F. Gutierrez.- Apress., 2018. – 532p.
9. Beaulieu A. Learning SQL: Master SQL Fundamentals / A. Beaulieu.- O'Reilly Media., 2008 – 338p.
10. Posta C. Microservices for Java Developers / C. Posta.- O`Reilly Media, inc., 2016 – 129 p.\
11. Baueur C. Java Persistence with Hibernate / C. Baueur.-Manning Publications., 2007 – 608p

12. Carnell J. Spring Microservices in Action / J. Carnell.- Manning Publications., 2017 – 384p